

7. Data Processing Subsystem (DPS) CSCIs

The software for the Data Processing subsystem is divided into:

- PRONG - Processing CSCI (Sections 7.1 and 7.2)
- SDPTK - SDP Toolkit CSCI (refer to 455-TR-001-001)
- AITTL - Algorithm Integration & Test CSCI (Section 7.4)

7.1 PRONG CSCI Overview

The Processing CSCI is responsible for the initiation, managing, and monitoring of the execution of science software algorithms. These science software algorithms are identified to the Processing CSCI through a PGE. From the ECS Glossary, a PGE is defined as "a set of one or more compiled binary executables and/or command language scripts; it is the smallest unit that can be scheduled for the Product Generation System (PGS, now the Planning and Data Processing Subsystems) processing." A PGE is equivalent to one processing job which requires the use of the Data Processing Subsystem's hardware and software resources. Generally, a PGE will be used for the generation of ECS data products, but a PGE may be defined to perform other types of processing, such as pre-processing of input data or the quality assurance processing of generated data products. PGEs that generate data products and perform quality assurance are provided by the Instrument Teams and algorithm developers. The Processing CSCI is informed of the required execution of a PGE through a data processing request message received from the Planning CSCI. The Processing CSCI will not initiate the execution of a PGE until all necessary data required as an input to the PGE is available, either on the local science processor, or within the IMF archive.

In support of the execution of the science software algorithms, the Processing CSCI has the following responsibilities:

- a. Manages the science software algorithm execution process.
- b. Manages science data processing computer hardware resources.
- c. Manages the flow of data required to execute a science software algorithm.
- d. Manages the flow of data produced by the execution of a science software algorithm.
- e. Provides an operational interface to allow monitoring of processing status, and manual intervention, when necessary, into science data processing operations environment, including processing queue control (e.g., DPR priorities).
- f. Provides an operational interface to support the quality assurance of generated data products.

7.1.1 CSCI Design Rationale

The Processing CSCI detailed design, as presented, is meant to provide a framework to build a robust science data processing system.

The Processing CSCI design rationale has been influenced by a set of design drivers.

- a. Separation of Planning and Processing—The roles and responsibilities of the Planning CSCI and Processing CSCI have been separated to support future evolving aspects of these functions.
- b. Data driven system—The Processing CSCI does not activate a data processing request until data is available to support the execution of the PGE associated with data processing request.
- c. Priority driven data processing—As a data processing request is input into the Processing CSCI, this data processing request has an assigned priority which is determined in the Planning CSCI.
- d. Exception handling—The capabilities to support recovery from the faults associated with the failure of a resource or a PGE are provided to support robustness in the science data processing environment.
- e. Extent of automation vs. manually supported operations—To support an efficient science data processing environment, a number of decisions have been made in the design to reflect the need to increase the amount of automated decision making required. This is driven by the need to support the generation of data products in an attended (by Operations staff) or unattended mode.
- f. Dynamic aspects of Planning and Processing—Both Planning and Processing software must have the capability to react to real-time events, i.e., PGE failures, resource failures, etc., which have affected the active production plan and the queue of data processing request jobs awaiting execution in Processing.

Each of these design drivers has impacted the framework of the Processing CSCI design. To emphasize their impact in the different functional areas affecting the design, a description of the design in certain areas and the underlying factors affecting the design has been provided. These descriptions are in the following sections:

- a. The Division of the Planning and Processing Subsystems
- b. Resource Management
- c. Quality Assurance
- d. Processing Error Architecture
- e. Processing/Planning Interface
- f. Processing/Data Server Interface (i.e., IMF interface for the Testbed)

The information provided in the following sections on the above listed topics is meant to summarize important aspects of the Processing CSCI detailed design.

7.1.1.1 The Division of the Planning and Processing Subsystems

The division of the Planning and Processing CSCIs was influenced by the following factors:

- a. Follows client/server architecture—Planning acts as the client, and Processing acts as the server. (Communication between the two subsystems is via a shared database instead of a message queue.) Planning is responsible for developing a production plan and informing Processing which activities must be executed as listed in the production plan.

- b. Increases fault tolerance of the Planning and Data Processing Subsystems—By separating the Planning services from the Processing services, an increase in the fault tolerance of the Planning and Data Processing Subsystem software is achieved. The failure of the Planning subsystem will not cause an immediate breakdown in production processing. Processing will be able to continue with production processing until the processing queues are depleted. Also, a failure of the Processing CSCI or processing hardware will not cause the Planning Subsystem to fail. Although the flow of information from Planning to Processing and Processing to Planning will be interrupted, this should not affect the overall production plan.
- c. Evolvability of the roles and responsibilities of Planning as ECS becomes operational—The separation of Planning services from Processing services allows for different configurations of Planning services and Processing services. For example, there may be some special event which would result in the Planning services of a DAAC creating a production plan for itself as well as other DAACs. By separating Planning from Processing, this becomes an easier problem to resolve.

Each of the above items has influenced the separation of Planning services from Processing services. This separation will support the general concept of the changing roles of Planning and Processing as ECS matures.

7.1.1.2 Resource Management

The Resource Management capabilities as presented in the Processing CSCI detailed design have been influenced by the following design drivers:

- a. Effective use of computer resources is required to support production processing—The Processing CSCI is responsible for the allocation of resources, i.e., disk space, memory, and CPU, to execute a PGE. To perform this role, Processing periodically tests the accessibility of the processing platforms and local storage devices.

The Processing CSCI allocates disk space, memory, and CPU to support the execution of a PGE. The PGE resource profile information for allocating disk space, memory and CPU are established during the AI&T time frame and will undergo updates as the science software matures in the science data processing environment

Before a PGE begins execution, all required resources must be available. This means that enough disk space and sufficient CPU must exist to support execution. These resources will be allocated to only support the execution of a given PGE. Also, as required through the Planning CSCI, all input data must be available before initiating the PGE. This will alleviate any deadlock situation where a PGE is awaiting some input data file or awaiting the use of a given resource currently in use by another PGE. Also, as determined through ECS dynamic modeling results, it seems that a majority of PGEs, as currently defined, will be executed individually on a CPU. This has been shown to provide an optimal solution. Even though this is the current direction, no Processing CSCI or COTS decision precludes the availability of the time-sharing of CPUs for multiple PGEs.

Also, during execution monitoring, Processing will monitor the use of the allocated disk space as a method of fault detection.

- b. Automated recovery from resource faults—Processing plays a limited role in resource fault recovery. Processing is responsible for protecting the state data maintained in the Processing software and for insuring that the execution of a PGE can resume upon the recovery of a resource. When a resource has failed, the Processing CSCI will update the resource management information associated with the failed resource to indicate that the resource can not be allocated for processing.
- c. Reporting of resource fault and other information—Processing supports fault reporting in two distinct ways:
 - 1. During PGE execution—While a PGE is executing, information is collected from the execution of the science software through the use of a Status Message File. Processing will inform the operator through the jobscape display and alarm manager screens. To recover from a resource fault, the Processing CSCI may determine if the PGE can be executed on another resource which is capable of supporting the execution of the PGE. This will be done by allowing the data processing request jobs to re-queue into the queue supporting the other resource. For a science software fault, Processing will use return code information provided by the PGE to determine the necessary steps to take for recovery purposes.
 - 2. Processing SW specific—if an event for which Processing is responsible (i.e., data staging, data destaging, execution environment monitoring) indicates a possible resource-specific fault. Processing here too will inform the operator through the jobscape display and the alarm manager.

7.1.1.3 Quality Assurance Monitoring

This section briefly describes quality assurance and the support provided by the PRONG CSCI.

- a. In-Line Q/A—This service is provided through the production environment. This is the automated quality assurance processing which can be provided by a PGE executing a quality assurance algorithm. At this time, a PGE could exist which only performs automated quality assurance on a generated product, or the quality assurance algorithm may be one part of a PGE which generates a data product. These quality assurance PGEs will be provided by instrument team algorithm developers.
- b. DAAC Manual Q/A—DAAC manual Q/A capabilities can be supported through the use of the IMF insert notification feature. The DAAC Q/A position would have to manually subscribe to the desired data product. The Data Server would provide notification to the DAAC Q/A monitor when a selected data product has been generated and is available for review. Also, this could allow DAAC Q/A to be de coupled from the generation and initial storage of the data product. DAAC Q/A activities would not necessarily have to occur when the product is generated.
- c. SCF Q/A—This service, while not provided by the Testbed, can be achieved through coordination with the DAAC personnel. An FTP poll area has been established in the Testbed where SCF personnel can take receipt of data products.

- d. User Q/A—User quality assurance activities occur as a data product is used to support research. The ability of the remote user to update quality assurance information is necessary. For existing data, a user can query for and request data directly.

By generating IMF insert notifications, the Processing CSCI lessens the dependency on man-in-the-loop activities affecting the generation of data products. Processing SW is being designed under the assumption that Data Processing must be able to generate products while in an attended or unattended mode. If the Q/A position is manned when the product is generated, the product can be reviewed while still on the Data Processing subsystem storage device. Normally, the Q/A position will request the product from the IMF archive.

The Processing CSCI will support an interface for visual display of a data product through the use of data visualization tool EOSView. Processing will also support an interface to allow for the update of the Q/A metadata which is associated with a data product.

The design approach was chosen for the following reasons:

- a. Based on the Data Processing design approach of allowing for manned and unmanned modes of operations.
- b. Follows Q/A approach for making products available for SCF review. No new paradigm to support DAAC manual Q/A has been introduced. This leads to maximum reuse of already developed software capabilities.
- c. Allows man-in-the-loop Q/A activities at the DAAC without serious impact on the processing resources. This approach would not require manual intervention before processing of other data products continues. If Q/A position is unmanned, the data products requiring Q/A review are stored in the IMF archive and await review.

7.1.1.4 Processing Error Architecture

This section provides a brief description of the error architecture approach adopted by the Processing CSCI and follows the common error handling approach adopted across all SDPS applications. Error architecture refers to the mechanisms used for error detection, reporting and recovery that are incorporated into the design of the Processing CSCI. It provides details on how the Processing CSCI will react when an error or exception, i.e., hardware or software, occurs during steady-state processing of a data processing request and the execution of a PGE.

This Processing CSCI error architecture approach was influenced by the following factors:

- a. Detection of different types of errors, i.e., science software (PGE) execution errors, Processing hardware errors, and Processing software errors.
- 1. The detection of science software errors which occur during the execution of a PGE are captured by the SDP Toolkit, and are propagated to the Processing CSCI through the use of a PGE return code status. The Processing CSCI will have knowledge of the error associated with a given return code, and as a result of this return code, may initiate a corrective measure, such as alerting the operations staff or preparing the PGE job for restart with updated diagnostic flags to initiate the capture of detailed diagnostic information. The definition of these return codes and the resulting corrective measures are defined in the white paper entitled “Establishing Science Software Exit Conditions for the Production Environment” (420-WP-006-002).

2. The detection of Processing HW and Processing SW errors is also a capability of SDPS.
- b. These errors must be reported to different user classes depending on the nature of the error, i.e., IT/algorithm developer, DAAC operations personnel. The IT/algorithm developer users are interested in the detailed error information associated with the execution of a PGE for the purposes of debugging a problem. This information is logged in status message files which are created and maintained during PGE execution by the SDP Toolkit. While the DAAC operations personnel are interested in whether a PGE was successfully executed, they are not as interested in the low-level details of the processing. In the event that a PGE has failed, the DAAC operations personnel will be alerted to the unsuccessful processing, and the alert information will be captured in the processing log for later use.
 - c. The recovery from an error will depend on the type of error. In almost all severe error cases, the recovery action will be to terminate the event which has caught the error. This approach is also required to support minimal human interaction by the operations staff. This helps support the concept of production processing occurring in an unattended mode.
 - d. Human interaction requirements to support production processing must be minimized. To support this, almost all errors will be logged and the operations staff will be alerted. Depending on whether production processing is occurring in an attended operations mode or unattended operations mode, the operations staff will have the capability to manually intervene to correct the error condition. Otherwise, the current activity will be terminated by Processing, and a new processing activity will be initiated.
 - e. Isolation of errors so as not to affect other Processing activities. science SW processing is terminated to isolate the cause of the error. Also, Processing HW may be taken off-line to correct the resource problems to support the isolation of the error condition.

7.1.1.5 Processing/Planning Interfaces

This section provides a brief description of the relationship between the Planning and Processing CSCIs.

The interface between the Planning and Processing CSCIs occurs through a common shared database, known as the PDPS database. The PDPS database is an RDBMS, Sybase, and contains all data which is persistent to applications associated with Planning, Processing, and Algorithm Integration and Test. The decision to share a common database was driven by the many common data structures which were apparent in the preliminary design specifications developed for the Planning and Data Processing Subsystems.

When a data processing request is planned for execution, as represented in the activated production plan which is created and maintained by Planning CSCI components, the knowledge of this data processing request is transferred into the Processing CSCI domain. This involves adding the definition of a series of jobs representing the data processing request to the COTS product being used to manage production by the Processing CSCI (i.e., AutoSys). Please see Section 7.1.3 for more information on the selected COTS product. As these jobs run, the Planning CSCI is capable of polling on the COTS product to retrieve status information for a given job.

7.1.1.6 Processing/MSS Interfaces

In support of system resource configuration management, a DAAC customized resource configu-

ration file allows the Processing CSCI to logically manage the allocation of resources to support science data production.

7.1.1.7 Processing/IMF Interfaces

The Processing CSCI has an interface to the IMF subsystem to support the staging (acquiring of data from the IMF subsystem) and destaging (insertion of data to the IMF subsystem) of data. The Processing CSCI requests the IMF subsystem to stage or destage data as required to support the generation of a data product. The Processing CSCI will inform the IMF subsystem where the data should be staged (what resource) or where the data should be destaged (copied) from. The IMF subsystem uses NFS to copy the data to the science processing resources to support staging of data and likewise uses NFS to copy the data from the science processing resources to support destaging of data. When the IMF subsystem has completed this task, the Processing CSCI will be informed whether or not the data has been staged or destaged successfully. The Processing CSCI is responsible for determining that data should be deleted from the science processing resources. When data is destaged, it is considered a copy operation, not a remove operation.

An additional interface exists for addressing the inspection and query of metadata for the purpose of acquiring and examining attributes of the data products which exist in the archive.

7.1.2 Processing CSCI Design Modifications Since CDR

1. A short time prior to CDR, each of the two CSCIs, Planning and Processing, were using their own instance of a Sybase database. This was later modified, due to the amount of redundant structures that were being defined. Both databases were merged into a single instance. In a similar vein, an analysis of the tables in the database revealed that two related yet distinct CSCs were making use of similar sets of tables. Further investigation lead to the discovery that both sets could be merged into a single set with only minor modifications to the tables and the object classes that interface with them. These CSCs, Resource Planning and Resource Management, now manipulate the set of database tables which comprise the resource model, but have different access methods owing to their heritage.
2. Due to delays in the delivery of the Science Data Server IF, this was replaced with the IMF IF, an additional CSCI introduced to minimally satisfy the archival needs of the PDPS. Not a client-server based entity, the IMF nonetheless satisfies the interfaces necessary to perform acquires (with detailed notification), inserts (of all required data types, e.g., PH), inspects, queries and updates of metadata.
3. As part of the Release A downsizing, the Subscription Manager CSC was replaced with the subscription reaper daemon. This only provides for insert notifications, which is the minimum required by the Pre-Release B Testbed.
4. To facilitate the processing of PGE error conditions, and the generation of the production history, an additional (7th) job, namely post processing, was added to the DPR job box. This new job is sequenced to begin immediately following the completion of the execution job. Since this new job interprets the error condition of the science PGE, failure of the execution job will not be witnessed until post processing has made this determination and has adjusted the state of the execution job accordingly.

5. To further clarify the state of the DPR as presented to the console operator, the DPR job box was defined to “fail” (i.e., turn RED) if any of the six administrative jobs in the job box fail. This effect will not preclude the restarting of the DPR, or the forced starting of an administrative job contained within (though automatic sequencing between administrative jobs may no longer hold, causing the user to force start one or more of the jobs within the DPR job box).
6. To better manage the disk space attached to the science processing platforms, staging directories are now created for any DPR which requires external data, unless all such data currently resides on the same science platform. This directory serves as the repository of data files (products and metadata) which are staged from the IMF archive. As data files are no longer required for processing, they are removed. The staging itself is removed when there are no more files contained within it.
7. To avoid deadlocks which can arise when two or more processes simultaneously access critical sections of software, a comprehensive resource locking class was introduced. For the Pre-Release B Testbed, this functionality was incorporated to control access to shared Resource Management objects. However, since this class only insures synchronous access to the PDPS software, and not to the lower-level Sybase software, an additional file locking class was introduced to guarantee this synchronous behavior down to the UNIX file system level.
8. Resource Management allocation of CPU and RAM is now maintained separately from that of the computer resource objects. This modification was necessary to maintain the integrity of the processor and memory allocations and deallocations. This is especially true in light of the fact that RAM allocations can affect the number of CPU allocations according to a ratio which is defined in the configuration parameter file for Resource Manager clients (e.g., the Execution Manager application).
9. In a measure to compensate for Autosys’ deficiencies in the area of resource management, an ingenious “file watching” mechanism was introduced to monitor, report, and correct for unanticipated file growth. This mechanism uses elements of both the AutoSys software and PDPS to effectively manage the science platform disk resources.

7.1.3 COTS Strategy

This COTS strategy section summarizes the objectives and technical approach which was taken to determine the optimal COTS solution required to support the job scheduling functions associated with the Processing CSCI management of the science processing resources. There were four objectives which influenced the decision on the type of COTS products selected;

- 1) Minimize custom code development
To insure an adequate solution for the Release A time frame, the COTS solution must minimize the amount of custom code development which was required to provide the complete Planning and Data Processing Subsystem software solution. This is necessary to mitigate the schedule risk associated with Release A which because of time constraints, will not support a large growth in custom code development.
- 2) Reach COTS decision to support Release A detailed design and implementation phase.

An early decision on the COTS product was required to insure that the required modifications to the Planning and Processing CSCI design could be made to support CDR and to ultimately support the Release A software development schedule.

3) Ease of integration into ECS software applications.

The selected COTS product must support command line or API style interfaces to support the extensive integration efforts which must occur to meet ECS requirements.

4) Scalability to Release B processing load.

The adopted technical approach consisted of the following steps:

- 1) Collecting information about the different types of COTS which could be used to support ECS Planning and Processing functions;
 - a)Vendor teleconferences and meetings
 - b)Customer teleconferences
 - c)Vendor site visits
- 2) Determining different types of software architecture given the COTS products and the unique ECS Planning and Processing requirements.
- 3) Analyzing the different classes of COTS packages to determine viability in ECS given the previously defined objectives.

The information gained in this process was used during the preparation of the RFP (Request For Proposal). This RFP was divided into mandatory, optional, and implementation features sections. The responses to these proposals were analyzed and scored based on the information provided by the vendor for each of these sections.

7.1.4 COTS Selection

The following sections provide information on the COTS products selected to support the Processing CSCI in performing management and monitoring activities associated with ECS' science data production environment. The product selected was Platinum Technology's AutoSys and AutoXpert products. A summary of AutoSys capabilities and a scenario which explains the set of actions taken to initiate a job is provided.

7.1.4.1 Platinum Technology's AutoSys

AutoSys is a job scheduling software application used to automate operations in a distributed UNIX environment. AutoSys performs automated job control functions required for scheduling, monitoring, and reporting on jobs that reside on any machine attached to a network. In ECS, the machines for which AutoSys will be responsible are the Science Processing and PDPS queueing hardware resources. In AutoSys, a job is defined as any UNIX command or shell script plus a variety of qualifying attributes which include conditions specifying when and where the job should be run.

AutoSys provides a complete system solution to support job scheduling. This includes an operator console, which allows human intervention into the AutoSys job stream, and provides various mechanisms for monitoring the AutoSys job stream. Provided with the interface are capabilities to view job definitions, change the state of a job, modify the job definition, and alter job dependency

information. Also provided with the operations interface is an alarm manager. The alarm manager can be used to assist in monitoring jobs and to react to fault situations. These alarms can be set through the definition of a job.

7.1.4.2 AutoSys Integration into the Processing CSCI Detailed Design

For the Processing CSCI, AutoSys provides the job scheduling engine for the Processing CSCI. The AutoSys event processor will manage all the events that occur in the science data production environment. The AutoSys database is a separate instance, apart from the PDPS database. For detailed design, the current assumption is that the AutoSys provided processes will manage the processing environment when the production facility is operating in a steady state manor. Once a production plan is activated (see Section 6), and the individual data processing requests (DPRs) begin to take form in AutoSys, AutoSys will begin managing and monitoring the execution of jobs combined therein.

To support the execution of jobs, AutoSys will require additional help in the following areas:

- a. **Resource Management**—Allocation of sufficient resources, i.e., disk storage, to support execution. Currently, AutoSys provides no mechanisms for managing disk storage. This is a potential enhancement that will be added to their product. Also, the monitoring of resources will not be done by AutoSys. This is a Processing responsibility.
- b. **Data Management**—Manages the staging, destaging, and retention of data on Processing resources.
- c. **PGE Execution Management**—Initiates and monitors execution of a PGE and the files it produces.

The following paragraphs briefly illustrates the operational concepts of the new Processing CSCI design. They discuss the following:

- a. How a job schedule is created
- b. How jobs are prepared and initiated, and
- c. How post-processing takes place.

These applications will be initiated by AutoSys at the job level. For each PGE, a series of preparation, execution, and post-execution jobs will be defined. The preparation jobs will manage the staging of data (if necessary), the allocation of resources to support execution, and preparation of the runtime environment. The execution job will be used to initiate and monitor the execution of the PGE. The post-execution job will be used to destage and deallocate resources, and retrieve the pertinent runtime statistics which will be used to generate a production history and satisfy the system's reporting needs.

To accomplish the set-up of these jobs, AutoSys' capabilities to create and manage job dependencies and to create job boxes, which consist of a series of related jobs, will be used. For each data processing request, which contains the data required to support the execution of one PGE, received from Planning, a corresponding job box, which contains a series of jobs to allocate resources, stage data, prepare the environment, execute the PGE, recover processing statistics, destage data, and deallocate resources, will be created. In Figure 7.1.4.2-1, the diagram shows the steps involved in providing job information to AutoSys. The Production Planning Workbench component of the Planning CSCI is the initiator of this activity. The Production Planning Workbench component will

use the Scheduler (DpPrScheduler) public class which was created, along with a few other classes, to provide an interface to AutoSys. This class encapsulates AutoSys defined capabilities and will manage the information flow from Planning to AutoSys. Through the use of API and command line interfaces and lower level classes, the Scheduler class provides job definitions to the AutoSys Database. Also, through the DpPrScheduler, the Planning CSCI will be able to request the status for the jobs associated with a data processing request.

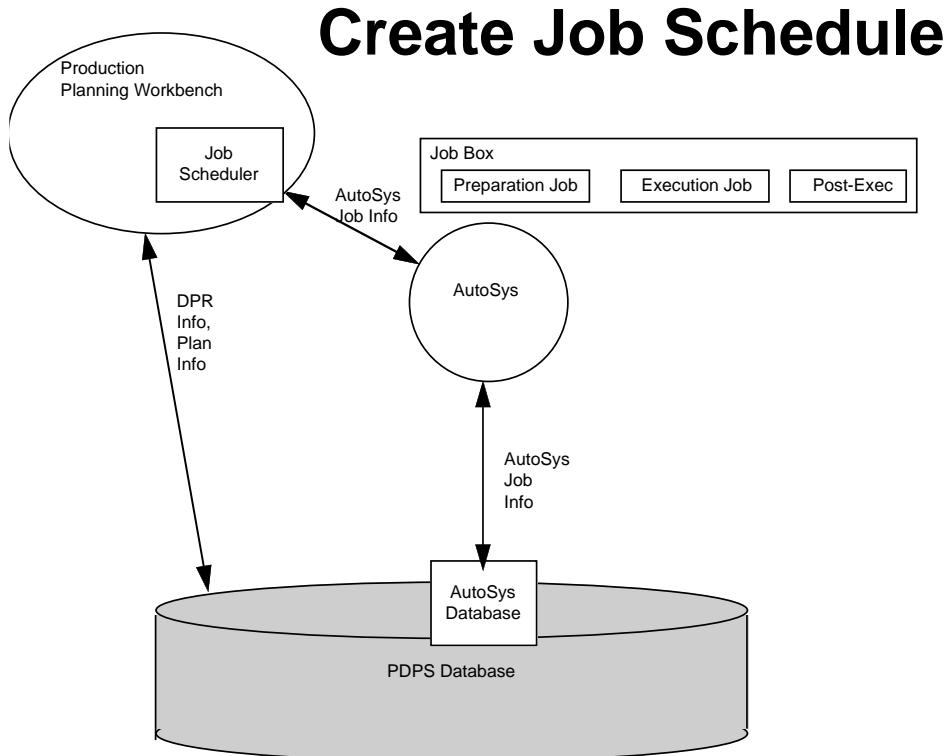


Figure 7.1.4.2-1. Scheduling Jobs using AutoSys

Each of these jobs actually consist of a command which initiates a process or processes to perform the task desired. The process will perform the desired functions and gracefully terminate. The successful completion of their task will result in the next dependent job being released until all jobs within the job box have completed. Besides managing normal operations through these mechanisms, failures which occur within a job box sometimes trigger special fault recovery procedures, e.g., redefine PGE executive job for DEBUG mode.

The Processing CSCI custom components will be initiated by AutoSys to perform support activities, such as resource allocation, staging and destaging data, and interfacing with the PGE. Figure 7.1.4.2-2 shows the series of steps and resulting actions performed by the Processing CSCI components.

Initiate Preparation Job

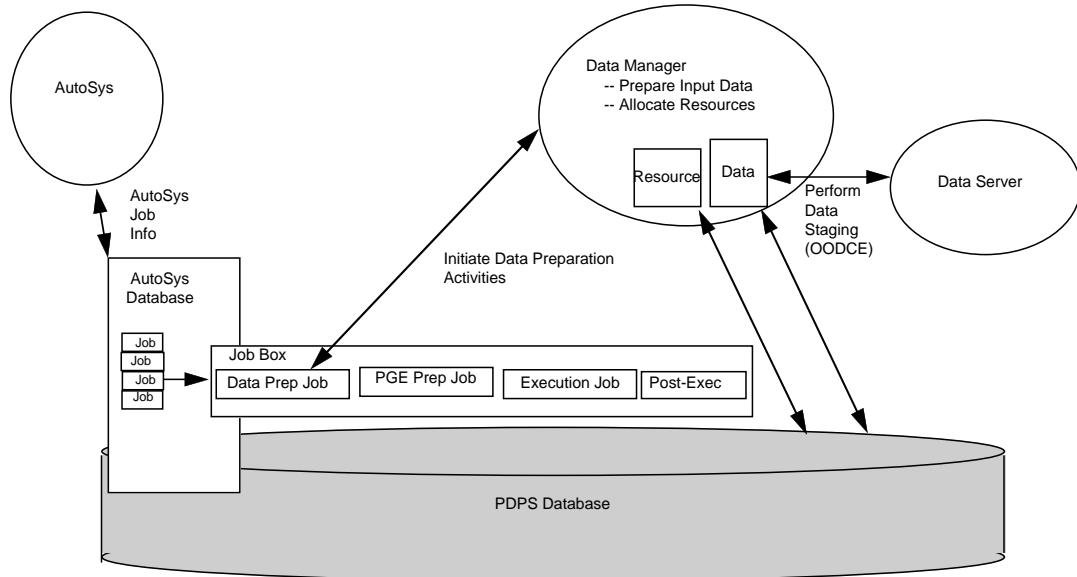


Figure 7.1.4.2-2. Initiating Processing Components using AutoSys

AutoSys also provides logging and report generation in order to capture information on job results and status changes.

7.1.4.3 Platinum Technology's AutoXpert

This product provides mechanisms to monitor and manage the job schedule which currently is being processed in AutoSys. This product is a GUI which provides different methods of viewing the progress of the job schedule, determining corrective measures when required, and providing capabilities to play what-if simulations to determine the affects of modifying the active job schedule. AutoXpert allows the job schedule to be represented at many different levels of abstraction. This concept is an important one given the large numbers of jobs expected to be executed per day at a given DAAC site. These abstract views include a time-line, gantt chart, and job data flows. As part of these views, jobs which are not following predicted behaviors will be color coded which will allow the operations staff to intervene, if necessary.

AutoXpert provides additional capabilities to perform simulations using the current active job schedule. These capabilities will assist the operations staff in judging the downstream effects of the failure of a job, the unavailability of a resource, the late arrival of data, and a job running longer than predicted. These capabilities are provided and can be used in real-time for projections, but also in a simulation mode, where the schedule can be sped up or other job information can be changed to judge what the impact would be. These additional capabilities will help assist DAAC operations

personnel in determining what impact production anomalies will have against the active plan and may be used to determine if a replan of production is necessary, though these scenarios may not apply to a particular DAAC's use of the Pre-Release B Testbed.

7.2 Subsystem Context

The Processing CSCI interfaces with the Planning and IMF subsystems to fulfill its responsibilities (see Figure 7.2-1):

- a. Planning Subsystem—The Planning Subsystem is responsible for creating a production plan for the Processing CSCI. The production plan information is conveyed to the Processing CSCI through the use of data processing requests. Each data processing request represents one processing job to be performed by a Data Processing Subsystem computer resource. Processing will provide status information to Planning to assist in production management activities of the Planning CSCI.
- b. IMF Subsystem—To support the creation of ECS data products, the Processing CSCI needs the capability of requesting and receiving data (Data Staging) from the IMF archives where the data is maintained. Also, the Processing CSCI needs the capability of transferring data (data destaging) to the IMF repository for the purpose of archiving a generated data product.
- c. Operations Interface—To support the management and monitoring of the execution of a PGE and the creation of ECS data products, a HMI interface is provided. This interface will provide services to support the collection of status for a data processing request, the cancellation, suspension/resumption and/or modification of a data processing request, and monitoring of the health of Data Processing Subsystem hardware resources. Also, this interface will be used to support manual quality assurance operations performed at the DAAC.
- d. SDP Toolkit Interface—To support PGE execution, the Processing CSCI provides information on the location of input data and the location of where the generated output data products are to be maintained. While a PGE is executing, the Processing CSCI monitors the execution of the PGE and informs the operations staff of current status. Status may include current processing event history (what is happening, i.e., data staging, execution). Also, monitoring will be needed to make sure that the processing activity is executing properly. Upon completion of the execution of a PGE, the Data Processing CSCI will inform Planning and will initiate the transfer of the generated data product (if necessary) to the IMF.

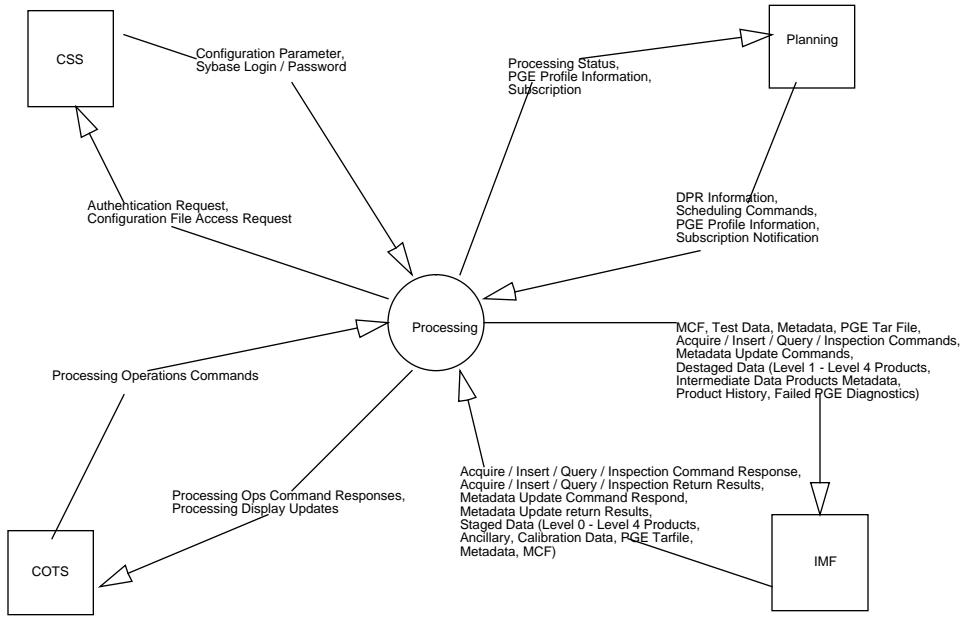


Figure 7.2-1. Processing CSCI Context Diagram

7.3 PRONG Object Model

The Processing CSCI Object Model is actually composed of a number of differing views of components which compose the Processing CSCI. These components provide different abstract views of the Processing CSCI design to assist in developing an understanding of the design.

Besides the above listed Processing CSCI components, the Processing CSCI is dependent on the PDPS Database for the management of Processing CSCI persistent data storage. More information on the PDPS Database is contained in the PDPS Database CSC of the Planning CSCI Design Specification.

7.3.1 Processing_Database_Interface Class Category

7.3.1.1 Overview

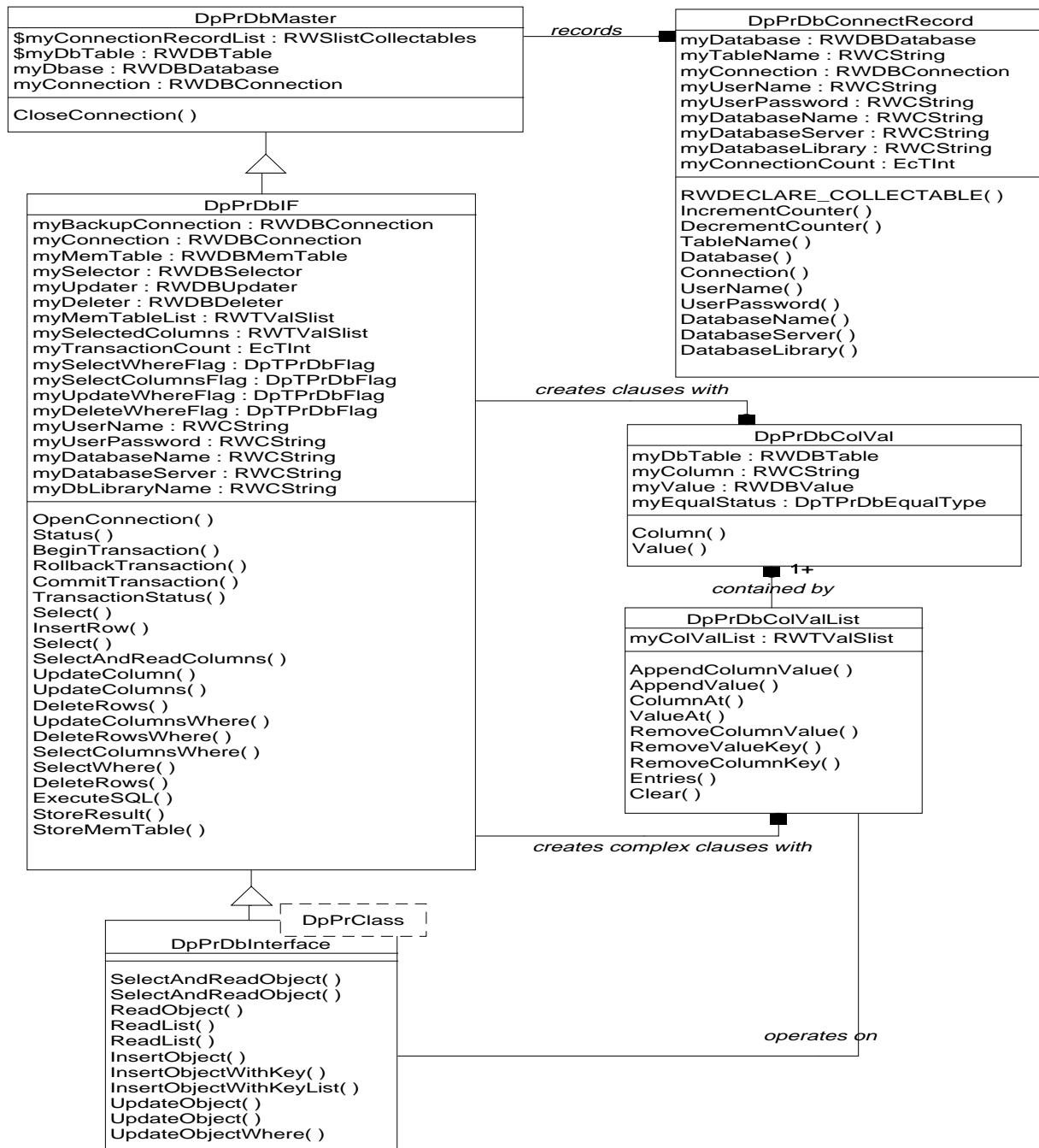


Figure 7.3.1.1-1 Database_Interface

7.3.1.2 Processing_Database_Interface Classes

7.3.1.3 DpPrDbColValList Class

Overview:

Export Control: **Public**

Inheritance Relationships:

Attributes:

myColValList: RWTValslist

Constructors and Destructor:

DpPrDbColValList();

This list will store column-value objects. Each item in this list is an object of class DpPrDbColVal which has column and value as its attribute.

DpPrDbColValList(const DpPrDbColValList& listObject);

virtual ~DpPrDbColValList();

Operations:

- AppendColumnValue

EcTVoid AppendColumnValue(const RWCString& columnName, const RWDBValue& value, DpTPrDbEqualType equalStatus);

This operation appends the specified column and value into the list object. If no value specified, it is NULL (rwdbNull by default).

- AppendValue

EcTVoid AppendValue(const RWDBValue& value, DpTPrDbEqualType equalStatus);

This operation appends only Value into the list. The corresponding ColumnName of this value is assumed to be NULL.

- Clear

```
EcTInt Clear();
```

This operation removes all items in the collection.

- ColumnAt

```
EcUtStatus ColumnAt(const RWDBValue& value, RWCString&  
returnColumn) const;
```

This operation returns the first column name that has value matches the specified value.

- Entries

```
EcTInt Entries() const;
```

This operation returns number of entries currently in the list.

- operator ==

```
EcTBoolean operator ==(const DpPrDbColValList& listObject);
```

- operator =

```
DpPrDbColValList& operator =(const DpPrDbColValList&  
listObject);
```

- operator []

```
DpPrDbColVal& operator [](EcTInt index);
```

- operator []

```
DpPrDbColVal operator [](EcTInt index) const;
```

- RemoveColumnKey

```
EcTInt RemoveColumnKey(RWCString& columnName, DpTPrDbRemoveType  
removeType);
```

This operation removes the first item in the list that has column name matches with the specified column name. If RemoveType is specified as DpEPrDbALL, then the operation will remove all items that matche the specified value. Number of items removed is returned.

- RemoveColumnValue

```
EcTInt RemoveColumnValue(const RWCString& columnName, const  
RWDBValue& value, DpTPrDbRemoveType removeType);
```

If RemoveType is not specified, by default this operation removes the first item in the list that matches the specified columnName AND value. If RemoveType is specified explicitly as DpEPrDbALL, then the operation will remove all the items in the list that have column name and value matches the specified ColumnName and Value. Return number of items removed.

- RemoveValueKey

```
EcTInt RemoveValueKey(const RWDBValue& value, DpTPrDbRemoveType  
removeType);
```

This operation removes the first item that has value matches with the specified key Value by default RemoveType. If RemoveType is specified explicitly as DpEPrDbALL, then the operation will remove all items that matches the specified value. Number of items successfully removed is returned.

- ValueAt

```
EcUtStatus ValueAt(const RWCString& column, RWDBValue&  
returnValue) const;
```

This operation returns the first value that has column name matches the specified column name.

7.3.1.3.1 DpPrDbColVal Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

myColumn: RWCString

myDbTable: RWDBTable

myEqualStatus: DpTPrDbEqualType

myValue: RWDBValue

Constructors and Destructor:

```
DpPrDbColVal();
```

```
DpPrDbColVal(const DpPrDbColVal& dbColVal);
```

Default destructor This copy constructor, the assignment operator and the equality operator are supported so that objects of this class can be stored in RWTValslist

```
virtual ~DpPrDbColVal();
```

Default constructor

Operations:

- Column

```
RWCString Column() const;
```

This operation returns the value of private attribute myColumn.

- GetColumnValue

```
EcTVoid GetColumnValue(RWCString& column, RWDBValue& value)  
const;
```

This operation returns the address of column and value retrieved from the value of privates attributes myColumn and myValue.

- GetCriterion

```
RWDBCriterion GetCriterion() const;
```

- GetCriterion

```
RWDBCriterion GetCriterion(const RWDBTable& dbTable) const;
```

- GetEqualStatus

```
DpTPrDbEqualType GetEqualStatus() const;
```

- operator ==

```
EcTBoolean operator ==(const DpPrDbColVal& dbColVal);
```

- operator =

```
DpPrDbColVal& operator =(const DpPrDbColVal& dbColVal);
```

Copy constructor

- SetColumnValue

```
EcTVoid SetColumnValue(const RWCString& column, const RWDBValue&
value, DpTPrDbEqualType equalStatus);
```

This operation sets the value of private attributes myColumn and myValue to the new values specified by Column and Value.

- SetColumn

```
EcTVoid SetColumn(const RWCString& column, DpTPrDbEqualType
equalStatus);
```

This operation sets the value of the private attribute myColumn to the new value specified by Column

- SetEqualStatus

```
EcTVoid SetEqualStatus(DpTPrDbEqualType equalStatus);
```

- SetValue

```
EcTVoid SetValue(const RWDBValue& value, DpTPrDbEqualType
equalStatus);
```

This operation sets the value of the private attribute myValue to the new value specified by Value.

- Value

```
Value() const;
```

This operation returns the value of private attribute myValue.

7.3.1.3.2 DpPrDbConnectRecord Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

```
myConnectionCount : EcTInt  
  
myConnection : RWDBConnection  
  
myDatabaseLibrary : RWCString  
  
myDatabaseName : RWCString  
  
myDatabaseServer : RWCString  
  
myDatabase : RWDBDatabase  
  
myTableName : RWCString  
  
myUserName : RWCString  
  
myUserPassword : RWCString
```

Constructors and Destructor:

```
DpPrDbConnectRecord();  
  
DpPrDbConnectRecord(const RWDBDatabase& database, const  
RWDBConnection& connection, const RWCString& userName, const  
RWCString& userPassword, const RWCString& databaseName, const  
RWCString& databaseServer, const RWCString& databaseLibrary);  
Alternate constructor.  
  
virtual ~DpPrDbConnectRecord();
```

Operations:

- Connection

```
Connection();
```

- DatabaseLibrary

```
RWCString DatabaseLibrary();
```

- DatabaseName

```
RWCString DatabaseName();
```

- DatabaseServer

```
RWCString DatabaseServer();
```

- Database

```
Database();
```

- DecrementCounter

```
EctInt DecrementCounter();
```

- GetCounter

```
EctInt GetCounter();
```

- IncrementCounter

```
EctInt IncrementCounter();
```

- RWDECLARE_COLLECTABLE

```
int RWDECLARE_COLLECTABLE(DpPrDbConnectRecord );
```

- SetConnection

```
EctVoid SetConnection(const RWDBConnection& connection);
```

- SetTableName

```
EctVoid SetTableName(const RWCString& tableName);
```

- TableName

```
RWCString TableName();
```

- UserName

```
RWCString UserName();
```

- UserPassword

```
RWCString UserPassword();
```

7.3.1.3.3 DpPrDbIF Class

Overview:

Export Control: **Public**

Inheritance Relationships:

Inherits from **DpPrDbMaster**

Attributes:

```
myBackupConnection: RWDBConnection
```

```
myConnection: RWDBConnection
```

```
myDatabaseName: RWCString
```

```
myDatabaseServer: RWCString
```

```
myDbLibraryName: RWCString

myDeleteter: RWDBDelete

myDeleteWhereFlag: DpTPrDbFlag

myMemTableList: RWTValslist

myMemTable: RWDBMemTable

mySelectColumnsFlag: DpTPrDbFlag

mySelectedColumns: RWTValslist

mySelector: RWDBSelector

mySelectWhereFlag: DpTPrDbFlag

myTransactionCount: Ec TInt

myUpdater: RWDBUpdater

myUpdateWhereFlag: DpTPrDbFlag

myUserName: RWCString

myUserPassword: RWCString
```

Constructors and Destructor:

```
DpPrDbIF();
```

Default constructor.

```
DpPrDbIF(const RWCString& userName, const RWCString&
userPassword, const RWCString& databaseName, const RWCString&
databaseServer, const RWCString& databaseLibrary);
```

This alternate constructor creates an interface object and make connection to database. Status() member function must be invoke to check for the validity of the object. If Status returns Ok, then this object is valid for any database manipulations. Otherwise, connection has not been established and any database interaction will cause failure result.

```
virtual ~DpPrDbIF();
```

This destructor orderly destroys the object and disconnect the connection to database.

Operations:

- BeginTransaction

```
EcUtStatus BeginTransaction();
```

Encapsulates begin transaction in SQL

- CommitTransaction

```
EcUtStatus CommitTransaction();
```

Encapsulates commit transaction in SQL

- DeleteRowsWhere

```
EcUtStatus DeleteRowsWhere(EcTInt& rowsDeleted, const
RWDBCriterion& whereClause);
```

This operation execute delete with complex where clause. In order to invoke this operation, applicatin must invoke SetTableForComplexWhere operation FIRST. This operation is used when deleting rows from table with complex where.

- DeleteRows

```
EcUtStatus DeleteRows(EcTInt& rowsDeleted, const RWCString&
tableName, const RWCString& whereColumn, const RWDBValue&
whereValue, DpTPrDbEqualType equalStatus);
```

Delete all the rows from the given table that satisfy SINGLE where codition. If operation completes successfully, Ok status is returned and number of rows deleted is assigned to rowsDeleted. Otherwise, rowsDeleted is assigned with -1 and failure status is returned to caller. If DeleteRows with complex where is desired, SetTableForComplexWhere and DeleteRowsWhere must be invoked respectively.

- DeleteRows

```
EcUtStatus DeleteRows(EcTInt& rowsDeleted, const RWCString&  
tableName, const DpPrDbColValList& whereColumnValueList);
```

Delete all the rows from the given table that satisfy MULTIPLE where conditions. Note that relationship between each where clause in whereColumnValueList is AND ONLY. If operation completes successfully, Ok status is returned and number of rows deleted is assigned to rowsDeleted. Otherwise, rowsDeleted is assigned with -1 and failure status is returned to caller. If DeleteRows with complex where is desired, SetTableForComplexWhere and DeleteRowsWhere must be invoked respectively.

- ExecuteSQL

```
EcUtStatus ExecuteSQL(EcTInt& rowsAffected, const RWCString&  
sqlCommand, <any> ...);
```

Executing raw SQL commands. If sqlCommand is select-related command, the result will be stored in resultList. If successed, number of rows will be returned. Otherwise, number of rows is -1 for failure status.

- GetConnectionStatus

```
DpTPrDbConnectionStatus GetConnectionStatus();
```

Return connection status of this object

- GetCriterion

```
EcUtStatus GetCriterion(const RWDBTable& table, const  
DpPrDbColValList& constraintList, RWDBCriterion& criterion);
```

Obtain criterion with given constrain list. This operation is used by member functions ONLY.

- GetIntValuesAt

```
EcUtStatus GetIntValuesAt(const RWCString& columnName, const  
RWTValslist& inputList, <any> ...) const;
```

This operation obtains a list of integer values (returnedIntList) belong to the given columnName in the inputList. inputList usually obtained from SelectAndReadColumns. Status of the process is returned.

- GetMaxNum

```
EcUtStatus GetMaxNum(const RWCString& tableName, const  
RWCString& columnName, EcTInt& maxNumber);
```

This operation obtains the maximum number in the given columnName from the given tableName.

- GetMemTablePtr

```
EcUtStatus GetMemTablePtr(const RWCString& tableName,
RWDBMemTable& dbMemTable, const DpPrDbColValList&
whereColumnValueList);
```

Get Memory Table that retrieve all records from appropriate table.

- GetMinNum

```
EcUtStatus GetMinNum(const RWCString& tableName, const
RWCString& columnName, EcTInt& minNumber);
```

This operation obtains the minimum number in the given columnName from the given tableName. Status of the process is returned.

- GetNumOfRows

```
EcUtStatus GetNumOfRows(EcTInt& rowsCount, const RWCString&
tableName);
```

This operation obtain number of rows currently exists in the given tableName. Status of the process is returned.

- GetSelectedColumns

```
EcUtStatus GetSelectedColumns(const RWCString& tableName, const
RWTValslist& selectedColumnList, <any> ...);
```

Obtain a list of select columns. This operation is used by member functions ONLY.

- GetStringValuesAt

```
EcUtStatus GetStringValuesAt(const RWCString& columnName, const
RWTValslist& inputList, <any> ...) const;
```

This operation obtains a list of string values (returnedStringList) belong to the given columnName in the inputList. inputList usually obtained from SelectAndReadColumns. Status of the process is returned.

- InsertRow

```
EcUtStatus InsertRow(const RWCString& tableName, const
DpPrDbColValList& insertedRow);
```

This operation performs insertion of a row into tableName with values stored in insertedRow.

- OpenConnection

```
EcUtStatus OpenConnection(const RWCString& userName, const
RWCString& userPassword, const RWCString& database, const
RWCString& databaseServerName, const RWCString& libraryName);
```

Establish connection for this database interface object. Application must supply the following parameters. Once connection is established, the connection stays until the object is destroyed.

- RollbackTransaction

```
EcUtStatus RollbackTransaction();
```

Encapsulates rollback transaction in SQL

- SelectAndReadColumns

```
EcUtStatus SelectAndReadColumns(const RWCString& tableName,
const RWTValslist& selectedColumnList, <any> ..., const
DpPrDbColValList& whereColumnValueList);
```

This operation selects desired column names from tableName with simple where query stored in whereColumnValueList. If selectedColumnList is empty, the operation selects all the columns from tableName, and stores all the rows selected in resultedList. This whereColumnValueList contains DpPrDbColVal objects, and each object in the list ONLY has AND relationship to each other. The equality between columns and values are set using AppendColumnValue function of DpPrDbColValList class. If operation successes, resultedList contains all the rows selected and caller must use methods provided in RWTValslist<T> class and DpPrDbColValList class to retrieved out these rows. Otherwise, resultedList will be empty and failure status is returned. Operation DpPrDbInterface::GetStringValuesAt(...) and DpPrDbInterface::GetIntValuesAt(...) can be used to retrieve a list of values belong to a particular column in the resultedList.

- SelectColumnsWhere

```
EcUtStatus SelectColumnsWhere(const RWTValslist&
selectedColumnList, <any> ..., const RWDBCriterion&
whereClause);
```

This operation supports complex query with whereClause given. This whereClause consists of DpPrDbColVal objects ANDed or ORed together. The equality between Columns and Values are set in this DpPrDbColVal object using one of its Set member function. If operation successes, resultedList should contains all the rows selected based on the query condition. Failure return results resultedList to be empty.

- SelectWhere

```
EcUtStatus SelectWhere(EcTInt& rowsSelected, const
RWDBCriterion& whereClause);
```

This operation supports complex query with whereClaus given. This whereClause consists of DpPrDbColVal objects AND or OR together. The equality between Columns and Values are set within this DpPrDbColVal object using one of its Set member function. If operation successes, number of rows selected is assign to rowsSelected. Otherwise, -1 is assigned to rowsSelected.

- Select

```
EcUtStatus Select(EcTInt& rowsSelected, const RWCString&
tableName, const RWCString& whereColumn, const RWDBValue&
whereValue, DpTPrDbEqualType equalStatus);
```

This Select operation selects from table tableName with single where clause. If whereColumn, whereValue is not specified, the operation selects all rows from table tableName. Successful completion of the selection results number of rows selected to be assigned to rowsSelected. If the process of operation fails, failure status is returned and rowsSelected is assigned to -1. If complex query is desired, SetTableForComplexWhere must be invoked and SelectWhere or SelectColumnsWhere must also be invoked.

- Select

```
EcUtStatus Select(EcTInt& rowsSelected, const RWCString&  
tableName, const DpPrDbColValList& whereColumnValueList);
```

Default is equal. This Select operation encapsulates the following SQL select * from tableName where column1 = value1 and column2 = value2 and so on... Note that the logical relationship between where column-values is AND ONLY. If whereColumnValueList is empty(no where clause), this implies the operation selects all rows from tableName. If the operation successes, number of rows selected is assigned to rowsSelected. Otherwise, -1 is assgined to rowsSelected and failure status is returned to caller. If complex query is desired, SetTableForComplexWhere must be invoked and SelectWhere or SelectColumnsWhere must also be invoked.

- SetTableForComplexWhere

```
EcUtStatus SetTableForComplexWhere(const RWCString& tableName);
```

This operation sets tableName so that complex where can be done for select, update, delete operations. This operation must be invoked before perform any complex where operations such as SelectColumnsWhere, SelectWhere, UpdateColumnsWhere, DeleteRowsWhere.

- Status

```
Status();
```

Return the status of this object

- StoreMemTable

```
EcUtStatus StoreMemTable(const RWDBMemTable& dbMemTable);
```

Store mem Table so that ReadList can retrieve records from approriate table. This operation is used by member functions ONLY.

- StoreResult

```
EcUtStatus StoreResult(const RWDBTable& dbResultTable, <any>  
...);
```

- TransactionStatus

```
DpTPrDbFlag TransactionStatus();
```

Return the status of transaction. If status return is DpEPrDbON, transaction has begun. Otherwise, transaction has not yet begun.

- UpdateColumnsWhere

```
EcUtStatus UpdateColumnsWhere(EcTInt& rowsUpdated, const  
DpPrDbColValList& assignColumnValueList, const RWDBCriterion&  
whereClause);
```

This operation execute update with complex where clause. In order to perform this operation, application must invoke SetTableForComplexWhere operation FIRST. This operation is used when updating with complex where clause ONLY.

- UpdateColumns

```
EcUtStatus UpdateColumns(EcTInt& rowsUpdated, const RWCString&  
tableName, const DpPrDbColValList& assignColumnValueList, const  
DpPrDbColValList& whereColumnValueList);
```

This operation updates rows from the given tableName with multiple where column-values. Note that relationship between each where clause in the list is AND ONLY. If there is no entries in whereColumnValueList, UpdateColumns will update ALL the rows in the table. Rows affected will be assigned to rowsUpdated. If process completes successfully, OK status is returned and number of rows updated is assigned to rowsUpdated. Otherwise, failure status is returned and rowsUpdate is assigned with -1. If complex where is desired for update, SetTableForComplexWhere and UpdateColumnsWhere must be invoked respectively.

- UpdateColumn

```
EcUtStatus UpdateColumn(EcTInt& rowsUpdated, const RWCString&  
tableName, const RWCString& assignColumn, const RWDBValue&  
assignValue, const RWCString& whereColumn, const RWDBValue&  
whereValue, DpTPrDbEqualType equalStatus);
```

This operation updates rows from the given tableName with single assign column-value and single where column-value. If NULL is desired for whereValue, whereValue must be set to DpCPrDbNull instead of (char*) NULL. Number of rows affected is assign to rowsUpdated. If complex where clause is desired, SetTableForComplexWhere and UpdateColumnsWhere operations must be invoked respectively. If operation completes successfully, rows updated is assigned to rowsUpdated and Ok status is returned. Otherwise, rowsUpdate is assigned with -1 and failure status is returned to caller.

7.3.1.3.4 DpPrDbInterface ParameterizedClass

Overview:

Export Control: Public

Inheritance Relationships:

Inherits from [DpPrDbIF](#)

Attributes:

Constructors and Destructor:

```
DpPrDbInterface(DpPrDbInterface& dbIFObject);
```

```
DpPrDbInterface(const RWCString& userName, const RWCString&
userPassword, const RWCString& databaseName, const RWCString&
databaseServer, const RWCString& databaseLibrary);
```

Default constructor.

```
DpPrDbInterface();
```

This operation instantiates an interface object. It does not connect to database until OpenConnection operation is invoked.

```
virtual ~DpPrDbInterface();
```

This operation performs an orderly clean up of interface objects. Any connection to database attached to this object will be automatically disconnected.

Operations:

- GetMemTable

```
EcUtStatus GetMemTable(const RWCString& tableName, RWDBMemTable&
memTable);
```

- InsertObjectWithKeyList

```
EcUtStatus InsertObjectWithKeyList(const RWCString& tableName,
const DpPrDbColValList& keyValueList, const DpPrClass&
insertedObject);
```

This operation supports an insertion of single object into table specified by TableName with key values which are not part of the class attributes. Status of the process is returned.

- InsertObjectWithKey

```
EcUtStatus InsertObjectWithKey(const RWCString& tableName, const
RWCString& keyColumn, const RWDBValue& keyValue, const
DpPrClass& insertedObject);
```

This operation supports an insertion of single object into table specified by TableName with key value which is not one of the class attributes. Status of the process is returned.

- InsertObject

```
EcUtStatus InsertObject(const RWCString& tableName, const  
DpPrClass& InsertedObject);
```

This operation supports an insertion of single object into table specified by TableName. In order to perform this operation, Application class needs a well-define operation Status of the process is returned.

- operator =

```
DpPrDbInterface& operator =(DpPrDbInterface& dbIFObject);
```

- ReadList

```
EcUtStatus ReadList(EcTInt& rowsRead, RWSlistCollectables&  
pointerList, const RWCString& tableName);
```

If tableName is not provided, this operation copies the entired row(s), resulted from the selection, into pointer-based DpPrPointerList. The DpPrPointerList must contain a list of pointers that point to instantiated default objects. Number of these objects is obtained from the Select operation. Number of objects successfully retrieved into this list is returned. If tableName is provided explicitly (recommended), all objects selected previously will be retrieved into this list.

- ReadList

```
EcUtStatus ReadList(EcTInt& rowsRead, <any> ..., const  
RWCString& tableName);
```

This operation copies the entired row(s) resulted from the selection into object-based DpPrObjectList, if tableName is not provided. The DpPrObjectList must contain a list of instantiated default objects. Number of these objects is obtained from the Select operation. Number of objects successfully retrieved into this list is returned. If tableName is provided explicitly (recommended), all objects selected previously will be retrieved into this list.

- ReadObject

```
EcUtStatus ReadObject(DpPrClass& retrievedObject, const  
RWCString& tableName);
```

If tableName is not provided, this operation copies data from the current row(s) resulted from the current selection into ClientObj. Again, since one object is supplied, data in the first row will be copied into this object. If tableName is provided explicitly(recommended), the first row in the specified table resulted from previous selection (not necessarily from the current selection) is retrieved into retrievedObject.

- SelectAndReadObject

```
EcUtStatus SelectAndReadObject(const RWCString& tableName,
DpPrClass& retrievedObject, const RWCString& whereColumn, const
RWDBValue& whereValue, DpTPrDbEqualType equalStatus);
```

This operation retrieves a record from the given table based on single where clause, and fills the retrievedObject with values from retrieved record. Status of the process is returned.

- SelectAndReadObject

```
EcUtStatus SelectAndReadObject(const RWCString& tableName,
DpPrClass& retrievedObject, const DpPrDbColValList&
whereColumnValueList);
```

This operation retrieves a record from the given table based on multiple where clauses, and fills the retrievedObject with values from retrieved record. Status of the process is returned.

- UpdateObjectWhere

```
EcUtStatus UpdateObjectWhere(EcTInt& objectsUpdated, const
DpPrClass& assignObject, const RWDBCriterion&
complexWhereClause);
```

This operation must be invoked after UpdateWithComplexWhere in order for the table to be affected. This operation should be used with complex where clause ONLY.

- UpdateObject

```
EcUtStatus UpdateObject(EcTInt& objectsUpdated, const RWCString&
tableName, const DpPrClass& assignObject, const
DpPrDbColValList& whereColumnValueList);
```

This operation supports the update of the table with the given object based on the whereColumnValueList. If one of the columns in the row cannot be updated, the update process is failed and zero should be returned in this case. If update process completed successfully, number of updated rows will be returned. Application class that uses the following UpdateObject operations must have a well-defined operation: friend DpPrDbColValList& operator<<(DpPrDbColValList& dbColValListObject, const DpPrClass& classObject); If complex where is desired for update an object in the table, then SetTableForComplexWhere and UpdateObjectWhere must be invoked, respectively.

- UpdateObject

```
EcUtStatus UpdateObject(EcTInt& objectsUpdated, const RWCString&
tableName, const DpPrClass& assignObject, const RWCString&
whereColumn, const RWDBValue& whereValue, DpTPrDbEqualType
equalStatus);
```

This operation supports the update of the table with the given object where WhereColumn equals to WhereValue. If one of the columns in the row cannot be updated, the update process is failed

and zero should be returned in this case. If update process completed successfully, number of updated rows will be returned. Application class that uses the following UpdateObject operations must have a well-defined operation: friend DpPrDbColValList& operator<<(DpPrDbColValList& dbColValListObject, const DpPrClass& classObject); If complex where is desired for update an object in the table, then SetTableForComplexWhere and UpdateObjectWhere must be invoked, respectively.

7.3.1.3.5 DpPrDbMaster Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

`myConnectionRecordList: RWSlistCollectables`

`myConnection: RWDBConnection`

`myDbase: RWDBDatabase`

`myDbTable: RWDBTable`

Constructors and Destructor:

Operations:

- CloseConnection

```
EcUtStatus CloseConnection(const RWCString& userName, const
                           RWCString& userPassword, const RWCString& databaseName, const
                           RWCString& databaseServer, const RWCString& databaseLibrary);
```

- GetConnection

```
EcUtStatus GetConnection(const RWCString& userName, const  
RWCString& userPassword, const RWCString& databaseName, const  
RWCString& databaseServer, const RWCString& databaseLibrary);
```

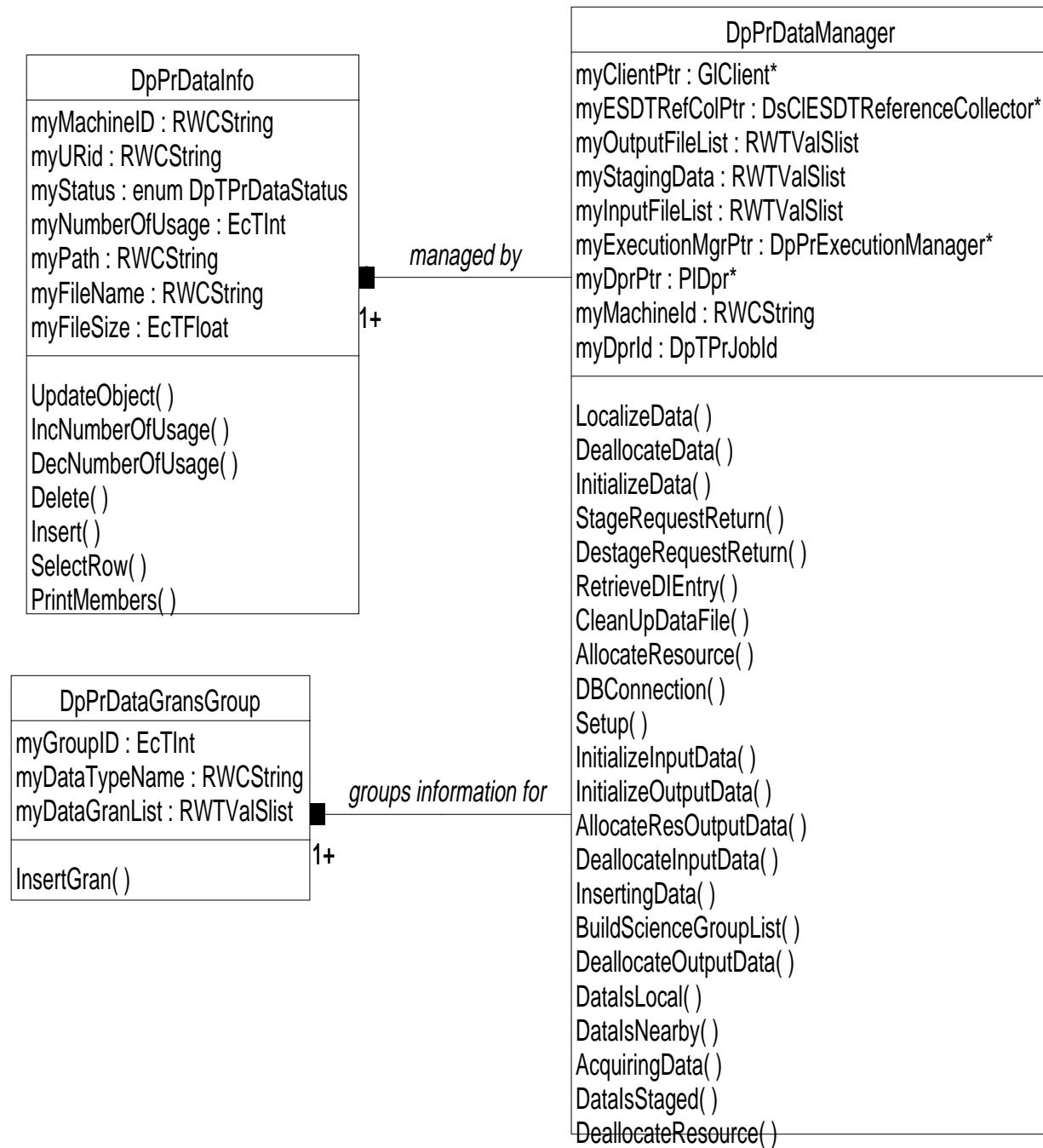
- `GetDbTable`

```
static RWDBTable GetDbTable();
```

- `SetDbTable`

```
EcTVoid SetDbTable(const RWDBTable& dbTable);
```

7.3.2 Processing_Data_Management Class Category



7.3.2.2 Processing_Data_Management Classes

7.3.2.3 DpPrDataGransGroup Class

Overview:

This class defines attributes and operations for a group of data granules based on groupId and DataType name.

Export Control: Public

Inheritance Relationships:

Attributes:

myDataGranList: RWTValslist

This is a list of data granules in a group

myDataTypeName: RWCString

An integer from 1 -> N This is the data type name of the group

myGroupID: EcTInt

class data members This is the group ID of a group of data granules (i.e. S1, Q1 ...)

Constructors and Destructor:

DpPrDataGransGroup(const DpPrDataGransGroup&);

The copy constructor for this class is provided in order to insert the object DpPrDataGransGroup into list of type RWTValSortedVector<T>

DpPrDataGransGroup(EcTInt groupId, const RWCString& dtype, const P1DataGranule&);

This is the alternate constructor

DpPrDataGransGroup();

This is the default constructor

~DpPrDataGransGroup();

This is the default destructor

Operations:

- **GetDataGranList**

```
const RWTValslist& GetDataGranList();
```

This public operation for other class(es) to use to get myDataGranList attribute value.

- **GetDataTypeName**

```
const RWCString& GetDataTypeName();
```

This public operation for other class(es) to use to get myDataTypeName attribute value.

- **GetGroupID**

```
const EcTInt GetGroupID();
```

This public operation is for other class(es) to use to get myGroupID attribute value.

- **InsertGran**

```
EcTVoid InsertGran(const PlDataGranule& );
```

This public operation inserts data granule object into myDataGranList

- **operator <**

```
RWBoolean operator <(const DpPrDataGransGroup& ) const;
```

This less than operator (<) is provided to allow the B-search to insert an object after all items that compare less than or equal to it, but before all items that do not.

- **operator ==**

```
RWBoolean operator ==(const DpPrDataGransGroup& ) const;
```

This equality operator supports the comparison of 2 DpPrDataGransGroup objects. If they equal, return TRUE; otherwise, return FALSE.

- **operator =**

```
DpPrDataGransGroup& operator =(const DpPrDataGransGroup& );
```

This assignment operator is provided to support the assignment of 2 or more DpPrDataGransGroup objects.

- **SetDataTypeName**

```
EcTVoid SetDataTypeName(const RWCString& dtype);
```

This public operation for other class(es) to use to set myDataTypeName attribute values.

- **SetGroupID**

```
EcTVoid SetGroupID(const EcTInt& groupID);
```

This public operation for other class(es) to use to set myGroupID attribute value.

7.3.2.3.1 DpPrDataInfo Class

Overview:

Since this class is a persistent class, it defines attributes and operations for manipulating the instances of objects as the entries in the Sybase DataBase. It can add, delete, update, and select the entry(ies) in the Sybase DataBase.

Export Control: **Public**

Inheritance Relationships:

Attributes:

myFileName: RWCString

File name of a particular data granule

myFileSize: EcTFloat

File size of a particular data granule

myMachineID: RWCString

class data members This indicates what machine this data granule locates on for a particular DPR.

myNumberOfUsage: EcTInt

A number indicates how many DPR use(s) this data granule in a day.

myPath: RWCString

This attribute indicates where, what path on a local machine that this data granule locates on.

myStatus: enum DpTPrDataStatus

The Status of data file

myURid: RWCString

The Identifier of an UR.

Constructors and Destructor:

DpPrDataInfo(const DpPrDataInfo& dataInfoObj);

default destructor Copy constructor, the assignment operator and the equality operator are provided so that objects of this class can be stored in RWTValSlist. This public copy constructor operation is provided in order to insert this object info object list of type RWTValSlist<T>.

DpPrDataInfo(RWCString , RWCString , RWCString , RWCString);

This is the alternate constructor of this class

```
DpPrDataInfo();
```

This is the default constructor

```
~DpPrDataInfo();
```

This is the default destructor

Operations:

- DecNumberOfUsage

```
EcTVoid DecNumberOfUsage();
```

A public operation for other class(es) to use to decrement NumberOfUsage attribute by 1.

- Delete

```
EcUTStatus Delete() const;
```

DataBase Interface operations This operation calls DeleteRows in DbInterface object to delete a row which represents this object in DATA_INFO table in DataBase. It emulates the Delete SQL statement: Delete from DATA_INFO where "whereCol" = "whereVal"

- GetFileName

```
RWCString GetFileName() const;
```

A public operation for other class(es) to use to get myFileName attribute value.

- GetFileSize

```
EcTFloat GetFileSize();
```

A public operation for other class(es) to use to get myFileSize attribute value.

- GetMachineID

```
RWCString GetMachineID() const;
```

A public operation for other class(es) to use to get MachineID attribute value.

- GetNumberOfUsage

```
EcTInt GetNumberOfUsage();
```

A public operation for other class(es) to use to get NumberOfUsage attribute value.

- GetPath

```
RWCString GetPath() const;
```

A public operation for other class(es) to use to get the Path attribute value.

- GetStatus

```
enum DpTPrDataStatus GetStatus();
```

A public operation for other class(es) to use to get the Status attribute value.

- GetURid

```
const RWCString GetURid();
```

A public operation for other class(es) to use to get URId attribute value.

- IncNumberOfUsage

```
EcTVoid IncNumberOfUsage();
```

A public operation for other class(es) to use to increment NumberOfUsage attribute by 1.

- Insert

```
EcUTStatus Insert();
```

This public operation adds a DataInfo row or entry into DATA_INFO table in DataBase. It emulates the Insert SQL statement.

- operator ==

```
EcTBoolean operator ==(const DpPrDataInfo& dataInfoObj);
```

This public equality operator supports the comparison of two DpPrDataInfo objects.

- operator =

```
DpPrDataInfo& operator =(const DpPrDataInfo& dataInfoObj);
```

This public assignment operator supports the assignment of 2 or more DpPrDataInfo objects.

- PrintMembers

```
EcTVoid PrintMembers();
```

- SelectRow

```
EcUTStatus SelectRow(const DpPrDbColValList& );
```

This public operation calls SelectAndReadObject in DbInterface to select one row or entry from DATA_INFO table in Sybase DB. This operation emulates the Select SQL statement: Select * from DATA_INFO where "wherocol" = "value" and ...

- SetFileName

```
EcTVoid SetFileName(const RWCString& fileName);
```

A public operation for other class(es) to use to set myFileName attribute value.

- SetFileSize

```
EcTVoid SetFileSize(EcTFloat fileSize);
```

A public operation for other class(es) to use to set myFileSize attribute value.

- SetMachineID

```
EcTVoid SetMachineID(const RWCString& machineID);
```

A public operation for other class(es) to use to set MachineID attribute value.

- SetNumberOfUsage

```
EcTVoid SetNumberOfUsage(EcTInt noOfUsage);
```

A public operation for other class(es) to use to set NumberOfUsage attribute value.

- SetPath

```
EcTVoid SetPath(const RWCString& path);
```

A public operation for other class(es) to use to set the Path attribute value.

- SetStatus

```
EcTVoid SetStatus(enum DpTPrDataStatus dataStatus);
```

A public operation for other class(es) to use to set the Status attribute value.

- SetURid

```
EcTVoid SetURid(const RWCString& urId);
```

A public operation for other class(es) to use to set the URid attribute value.

- UpdateObject

```
EcUtStatus UpdateObject(const DpPrDbColValList& );
```

This operation calls UpdateObject in DbInterface object to update field(s) value of a particular row in DATA_INFO table in DataBase.

7.3.2.3.2 DpPrDataManager Class

Overview:

This class defines attributes and operations for initializing and managing data granules required by a PGE during its execution. It also ensures data availability before the execution of a PGE. In case of input data is unavailable at our local system disk, it sends request to DataServer to stage data at a specific location that allocated by Resource Management. At the end of the successful execution of a PGE, it asks DataServer to destage (archive) output data and asks Resource Management to deallocate the sources.

Export Control: **Public**

Inheritance Relationships:

Attributes:

myClientPtr: G1Client*

= Data members This attribute contains an address of data server Client

myDprId: DpTPrJobId

This is DPR ID

myDprPtr: PlDpr*

This attribute contains an address of a DPR object

myESDTRefColPtr: DsClESDTReferenceCollector*

This attribute contains an address of DsClESDTReferenceCollector

myExecutionMgrPtr: DpPrExecutionManager*

This attribute contains an address of ExecutionManager object

myInputFileList: RWTValslist

A list of DataInfo entries for input

myMachineId: RWCString

This indicates the machineID that DPR is running on

myOutputFileList: RWTValslist

A list of DataInfo entries for output.

myStagingData: RWTValslist

A list of DataInfo entries that are being staged at DataServer.

Constructors and Destructor:

DpPrDataManager(EcUtStatus& status, const RWCString& dprId);

This is the other alternate constructor for this class.

DpPrDataManager(EcUtStatus& status);

This is the alternate constructor for this class.

DpPrDataManager();

= Public operations This is the default constructor

~DpPrDataManager();

This is the default destructor

Operations:

- AcquiringData

```
EcUtStatus AcquiringData(const RWCString& urId, DpPrDataInfo&  
dInfoObj);
```

This private operation is invoked to connect to DataServer and stage data from Data Server to local disk at the location indicated by Resource Manager.

- AllocateResource

```
EcUtStatus AllocateResource(<any> ...);
```

= Private operations This private operation is called to invoke Resource Manager to allocate resource for an (input/output) file.

- AllocateResOutputData

```
EcUtStatus AllocateResOutputData();
```

This private operation is called by LocalizeData to allocate resources for output data files

- BuildScienceGroupList

```
EcUtStatus BuildScienceGroupList(const PlDataGranule&  
dataGranObj);
```

This private operation is called to build the science group list

- CleanUpDataFile

```
EcUtStatus CleanUpDataFile();
```

A private operation to use to clean up unused DataInfo entries.

- DataIsLocal

```
EcUtStatus DataIsLocal(const DpPrDbColValList& whereList,  
DpPrDataInfo& dInfoObj, const RWCString& urId);
```

This private operation is invoked by LocalizeData operation perform necessary functions when data is at local disk.

- DataIsNearby

```
EcUtStatus DataIsNearby(const RWCString& urId, EcTFloat  
fileSize, const DpPrDataInfo& dInfoObj);
```

This private operation is invoked by LocalizeData operation to copy data from nearby local disk to this local disk.

- DataIsStaged

```
EcUtStatus DataIsStaged(const RWCString& urId, EcTFloat  
fileSize, PlDataGranule& dataGranule, DpPrDataInfo& dInfoObj);
```

This private operation is invoked by LocalizeData operation to stage data from Data Server to local disk at the location indicated by Resource Manager.

- DBConnection

```
EcUtStatus DBConnection();
```

This private operation is called by DpPrDataManager alternate ctor to make the connection to our database.

- DeallocateData

```
EcUtStatus DeallocateData();
```

After a PGE is completed, this operation is called (with DPRid) to decrement the NumberOfUsage by 1 for each data input granule indicates 1 less PGE uses this data, but it is not necessary to physically delete from the disk until we run out resources. Then it sends request to DataServer to destage (archive) output data files and the production history log file.

- DeallocateInputData

```
EcUtStatus DeallocateInputData();
```

This private operation is called by DeallocateData to decrement the NumberOfUsage count of input files by 1 indicates that this data file is 1 less used.

- DeallocateOutputData

```
EcUtStatus DeallocateOutputData();
```

This private operation is called by DeallocateData to interface with Data Server to destage (archive) output files to Data Server

- DeallocateResource

```
EcUtStatus DeallocateResource(const DpPrDataInfo& dInfoObj);
```

A private operation performs the following tasks: 1) Use system call to remove the data file physically at local host 2) call Resource Manager to deallocate resource 3) Delete the DataInfo entry from DataBase

- DestageRequestReturn

```
void DestageRequestReturn();
```

This operation is called by CallBack after a destage request was sent to DataServer. If output data is successfully archived, then it goes through all output data entries, deallocate source, and remove them from Data_Map table in DataBase.

- InitializeData

```
EcUtStatus InitializeData(const PlDpr& dpr);
```

This public operation is called to initialize data in the Database. When a DPR is first created, this gets called. It goes to each of Data granule and gets a URid. Then it searches through the DATA_INFO table in the Database to find the URid that matches the keyed UR. If it cannot find any entry, then it will create an entry and put it in the DataBase with NumberOfUsage field set to 1 (i.e. there is 1 DPR that needs this data granule). As it goes along, if it finds an entry in DataBase that matches the searched URid, then it increments the NumberOfUsage field by 1 (i.e. there is one more DPR that needs this data granule).

- InitializeInputData

```
EcUtStatus InitializeInputData(const PlDpr& dpr);
```

This private operation is called by InitializeData to initialize input data files.

- InitializeOutputData

```
EcUtStatus InitializeOutputData(const PlDpr& dpr);
```

This private operation is called by InitializeData to initialize output data files.

- InsertingData

```
EcUtStatus InsertingData(const RWTValSortedVector&  
sciGroupList);
```

This private operation is called to build the insert command by groups to interface with DataServer to destage (archive) output files to Data Server

- LocalizeData

```
EcUtStatus LocalizeData();
```

This operation mainly initializes and ensures availability of data before PGE is executed. First, it ensures that the required data for execution is available at our local disk. If it is not located on our local system disk, it checks to see if this data is already located on a nearby local system disk. If it is, then copy from nearby local disk to this local disk. If it is still not out there anywhere, it then asks the Resource Management to allocate some disk space on local disk and sends request to Data Server to stage data on local disk at the location indicated by Resource Management.

- PerformFTP

```
EcUtStatus PerformFTP(const DpPrDataInfo& nearbyDInfoObj, const  
DpPrDataInfo& localDInfoObj);
```

This private operation is invoked to copy data file from nearby local disk to this local disk by using FTP Object.

- RetrieveDIEntry

```
EcUtStatus RetrieveDIEntry(DpPrDataInfo& dInfoObj);
```

A public operation for other class(es) to use to get an entry of DpPrDataInfo, when a DATA_INFO entry is provided.

- Setup

```
EcUtStatus Setup();
```

This private operation is called by DpPrDataManager alternate ctor to get all necessary parameters from the application config file.

- StageRequestReturn

```
void StageRequestReturn();
```

This operation is called by CallBack after a stage request was sent to Data- Server. If data is successfully staged, then it goes through all staging data entries, updates the Status to LOCAL.

- UpdateMasterRecord

```
EcUtStatus UpdateMasterRecord(const RWDBValue& urId);
```

This private operation is called to decrement the NumberOfUsage count of master record by 1. The master record is the record that keeps track how many PGE needs this data file (regardless of MachineID) across multiple machines. This record has fileName as the primary key and machine Id as the secondary key, and machine Id is ALWAYS = NULL.

7.3.3 Processing_Exception Class Category

7.3.3.1 Overview

DpPrInterrupt
\$myState : EcTBoolean
State()
\$Handler()

7.3.3.2 Processing_Exception

7.3.3.3 DpPrInterrupt

Overview:

This class manages PRONG administrative job's process interrupt state. It manipulates the process's interrupt signal handling characteristics to provide a transparent mechanism for a running process to perform a graceful shutdown when requested.

Export Control: Public

Inheritance Relationships:

Attributes:

myState: EcTBoolean

This private static data member contains the interrupt state.

Constructors and Destructor:

```
DpPrInterrupt();
```

This is the constructor.

```
virtual ~DpPrInterrupt();
```

This is the destructor.

Operations:

- Handler

```
static EcTVoid Handler();
```

This private static function is the interrupt signal handler.

- State

```
EcTBoolean State();
```

This public member function returns the current interrupt state.

7.3.4 Processing_Execution_Management Class Category

7.3.4.1 Overview

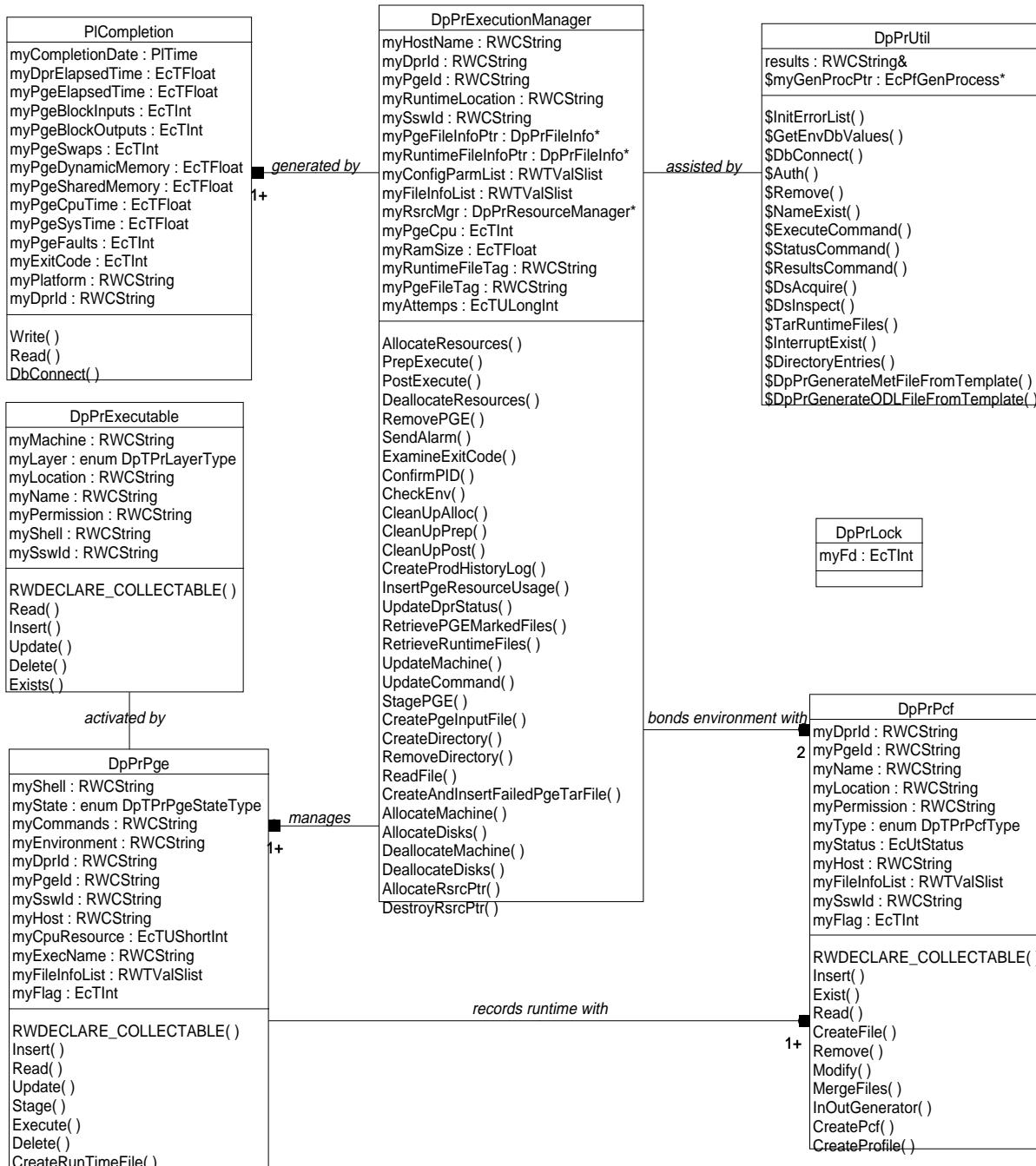


Figure 7.3.4.1-1 Execution_Management

7.3.4.2 Processing_Execution_Management Classes

7.3.4.3 DpPrExecutable Class

Overview:

Export Control: **Public**

Inheritance Relationships:

Attributes:

myLayer: enum DpTPrLayerType

The layer of the object can be used to determine if it requires direct execution on the part of the Processing System, or if it is indirectly executed from the PGE itself.

myLocation: RWCString

The disk location where the executable resides will be determined initially, but may be modified to reflect a change in resource allocations.

myMachine: RWCString

RWCString myPgeId; This defines the required machine and operating system combination required to execute this process. A null value indicates that the entity is capable of being run on any platform. This value will be used to determine alternate resources for execution if the initially allocated resource ever fails.

myName: RWCString

The actual executable or Status Message File (SMF) name is identified by this attribute.

myPermission: RWCString

The system permission settings may need to be set by the Processing System following the staging of the executable file or Status Message File (SMF).

myShell: RWCString

For objects which are shell scripts, as is expected for the main PGE, this shell may need to be explicitly invoked as part of the job command that the COTS Scheduler issues. This attribute does not apply to binary executables and Status Message Files (SMFs).

mySswId: RWCString

Science software Id which uniquely identifies the PGE software package

Constructors and Destructor:

```
DpPrExecutable(const DpPrExecutable& execObject);
```

This copy constructor is used to create a new object which have the same attribute as the old object. This operation is implemented so that object of this class can be stored in RWValSlist<T> An existing executable object. A new executable object is created with the same attribute as the old executable object.

```
DpPrExecutable(const RWCString& sswId, const enum
DpTPrLayerType& layer, const RWCString& machine, const
RWCString& name);
```

This operation constructs DpPrExecutable object so that this object can be used to retrieve persistent object from database table. Use Read to perform actual retrieval of this DpPrExecutable object. sswId: Id of the PGE in which this executable file is belong. name: Name of the executable file. location: Location the executable file. A DpPrExecutable object is initialized.

```
DpPrExecutable(EcUtStatus& returnStatus, const RWCString& sswId,
const RWCString& execName, const RWCString& execLocation, enum
DpTPrLayerType execLayer, const RWCString& execAccess, const
RWCString& machine, const RWCString& execShell);
```

This operation will initialize name, target, location, layer, and shell of the executable file. The operation signature includes name, target, location, layer, and shell of the existent executable file. An object of type DpPrExecutable is created to monitor the executable file.

```
DpPrExecutable();
```

This default constructor operation construct an empty executable object with unknown attributes. A default object of type DpPrExecutable is created.

```
virtual ~DpPrExecutable();
```

This destructor will perform an orderly cleanup and deletion of the object.

Operations:

- Delete

```
EcUtStatus Delete();
```

Delete record in EXECUTABLE table and remove PGE software from production machine. Disk space is deallocated.

- Exists

```
EcUtStatus Exists(EcTBoolean& exist);
```

This method checks for the existence of EXECUTABLE record. This method should be invoked be for Read method. Existence of the record is returned in output parameter.

- GetLayer

```
enum DpTPrLayerType GetLayer();
```

The Layer attribute is returned to the calling process.

- GetLocation

```
RWCString GetLocation();
```

The current directory path, as defined by the Location attribute, is returned to the calling process.

- GetMachine

```
RWCString GetMachine();
```

The value of the private attribute myMachine is returned.

- GetName

```
RWCString GetName();
```

The filename of the object, as defined by the Name attribute, is returned to the calling process.

- GetPermission

```
RWCString GetPermission();
```

The system permission settings, as defined by the Permission attribute, are returned to the calling process.

- GetPID

```
RWCString GetPID();
```

Temporarily uses myShell as Process ID.

- GetShell

```
RWCString GetShell();
```

The value of the Shell attribute is returned to the calling process.

- GetSswId

```
RWCString GetSswId();
```

This operation returns the value of the private attribute mySswId to caller.

- Insert

```
EcUTStatus Insert();
```

This operation inserts DpPrExecutable object into database. A record is inserted into EXECUTABLE table.

- operator ==

```
EcTBoolean operator ==(const DpPrExecutable& execObject);
```

This equality operator compare two DpPrExecutable objects. If they are equal, the return is EcDTrue. Otherwise return EcDFalse. This operation is implemented mainly so that objects of this class can be stored in the list of type RWValSlist<T>. Two existing executable objects.

None.

- operator =

```
DpPrExecutable& operator =(const DpPrExecutable& execObject);
```

This assignment operation is implement so that objects of this class can be stored in the list of type RWValSlist<T> An existing executable object. A new executable object which has the same value of the private attributes as the old executable object.

- Read

```
EcUtStatus Read();
```

This operation retrieves DpPrExecutable object from database table into the constructed object. A persistent object from database is retrieved into DpPrExecutable object.

- RWDECLARE_COLLECTABLE

```
int RWDECLARE_COLLECTABLE(DpPrExecutable );
```

- SetLayer

```
EcTVoid SetLayer(enum DpTPrLayerType layer);
```

Set private myLayer with given layer.

- SetNewLocation

```
EcUtStatus SetNewLocation(const RWCString& execNewLocation);
```

To cover the possibility that initially provided disk resources may need to be reallocated, this operation will allow for the update of the objects physical location. A new location name must be supplied. The file should be at the new location and database table should be updated.

- SetPID

```
EcTVoid SetPID(const RWCString& spid);
```

- Update

```
EcUtStatus Update();
```

When PGE is in staging progress, state (layer) of this object is STAGING to indicate other DPRs, which depend on this PGE, that this PGE is in STAGING progress (don't stage again),

and proceeds after the STAGING completed. This operation sets the state of this executable in database so that other process can detect the state and performs appropriate action.

7.3.4.3.1 DpPrExecutionManager Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

myAttempts: EcTULongInt

Number of attempts of checking for existence of PGE.

myConfigParmList: RWTValslist

This private attribute contains a list of parameters to be used by Execution Manager.

myDprId: RWCString

Data processing request Id.

myFileInfoList: RWTValslist

This private attribute contains a list of file info objects which are used to clean up resources.

myHostName: RWCString

Machine where job is running.

myPgeCpu: EcTIInt

This private attribute stores number of CPUs needed to process a PGE.

myPgeFileInfoPtr: DpPrFileInfo*

This attribute contains runtime file info.

myPgeFileTag: RWCString

This private attribute stores the name of PGE file tag used to perform disk resource allocation for PGE files.

myPgeId: RWCString

PgeId defined at SSI&T.

myRamSize: EcTFloat

This private attribute stores size of RAM needed to process a PGE.

myRsrcMgr: DpPrResourceManager*

This private attribute is a pointer to DpPrResourceManager object. It will be allocated a space when needed.

myRuntimeFileInfoPtr: DpPrFileInfo*

This attribute contains pge file info.

myRuntimeFileTag: RWCString

This private attribute stores the name of runtime file tag used to perform disk resource allocation for runtime files.

myRuntimeLocation: RWCString

Runtime directory where Process Control file and .profile is located.

mySswId: RWCString

Science software Id defined at SSI&T.

Constructors and Destructor:

DpPrExecutionManager(EcUtStatus& returnStatus);

This default constructor constructs default object with data members initialized to 0. It makes connections to database for interface objects for DpPrPge, DpPrPcf, and DpPrExecutable. Status of the process is returned.

virtual ~DpPrExecutionManager();

This destructor will effect the orderly cleanup and deletion of the manager object.

Operations:

- AllocateDisks

EcUtStatus AllocateDisks();

This private operation provides interface to Resource Manager to allocate disk resources.

- AllocateMachine

EcUtStatus AllocateMachine();

RM interface operations. This private operation provides interface to Resource Manager to allocate machine resource.

- AllocateResources

EcUtStatus AllocateResources(const RWCString& dprId);

The resource allocations required to support the running of a PGE are performed by this operation. Initial runs of a PGE on a specified Machine will trigger the allocation of disk resources required to store the associated executables and runtime Status Message Files (SMFs), and will initiate the staging of these files to the allocated locations. This operation also checks for the existence of PGE on the target machine. If PGE exists, job proceeds normally. Otherwise, disk allocation and PGE staging from data server will be performed. Each activation of this operation will trigger the allocation of processing resources for the specified Job run. Id of the job or Data processing request id is given. Resources for this DprId is allocated.

- AllocateRsrcPtr

```
EcUtStatus AllocateRsrcPtr();
```

This private operation allocates space for DpPrResourceManager pointer defined as private data member. Resource lock is enable.

- CheckEnv

```
EcUtStatus CheckEnv();
```

Checks for the availability of configuration parameters used by EM

- CleanUpAlloc

```
EcTVoid CleanUpAlloc();
```

Performs clean up of previously allocated resources. This method is normally invoked when interrupt signal occur.

- CleanUpPost

```
EcTVoid CleanUpPost();
```

Cleans up files, records and/or resources that have done for Post-execution. None.

- CleanUpPrep

```
EcTVoid CleanUpPrep();
```

Cleans up files/records/resources that have been done in Prep-execution.

- ConfirmPID

```
EcUtStatus ConfirmPID();
```

Confirm PID of EXECUTABLE record in PGE record.

- CreateAndInsertFailedPgeTarFile

```
EcUtStatus CreateAndInsertFailedPgeTarFile();
```

Creates the failed PGE tar file and metadata and inserts to DSS

- CreateDirectory

```
EcUtStatus CreateDirectory();
```

This private operation creates directory allocated by AllocateDisk, and inserts this directory into PCF table and/or EXECUTABLE table.

- CreatePgeInputFile

```
EcUtStatus CreatePgeInputFile(const RWTValslist& infoList, const  
RWCString& fullInfoFile, const RWTValslist& fileInfoList);
```

This operation creates input file to be read by the PGE wrapper. infoList - information contains commands to invoke the Toolkit shell. fileName - name of the input file. fileLocation - location of the input file. fileInfoList - contains a list of file names to be watched A file will be created at production machine in runtime directory.

- CreateProdHistoryLog

```
EcUtStatus CreateProdHistoryLog(const RWCString& fullProdName,  
const RWCString& fullLogName);
```

Creates production history log file. Since the format of this file is not known, resource usage information will be written to production history file for now. fullProdName - full path name of the production history file. fullLogName - full path name of log file written by the PGE wrapper to report resource usage information.

- DeallocateDisks

```
EcUtStatus DeallocateDisks();
```

This private operation provides interface to Resource Manager to deallocate disk resources.

- DeallocateMachine

```
EcUtStatus DeallocateMachine();
```

This private operation provides interface to Resource Manager to deallocate machine resources.

- DeallocateResources

```
EcUtStatus DeallocateResources(const RWCString& dprId, enum  
DpTPrEMCondition condition);
```

The processing resources allocated for a particular run of the PGE, as specified by the dprId identifier, will be released. All disk resources associated with the PGE remain active. Data processing request Id is required for input. All resources associated with this DprId will be deallocated.

- DestroyRsrcPtr

```
EcTVoid DestroyRsrcPtr();
```

Destroys DpPrResourceManager object. Memory for this object is deallocated. Resource lock is released.

- ExamineExitCode

```
EcUtStatus ExamineExitCode(EcTInt exitCode, RWCString& exitMessage);
```

Examine exit code, then determine appropriate action. Also return the corresponding message

- GetDprId

```
RWCString GetDprId();
```

The value of the private attribute myDprId is returned.

- GetHistoryFile

```
EcUtStatus GetHistoryFile(const RWCString& dprId, RWCString& historyFile);
```

This method retrieves production history file record stored in PCF table. This is mainly used by Data Manager. DprId - Data processing request ID. Full path name of history file is return in output parameter.

- GetJobState

```
EcUtStatus GetJobState(const RWCString& dprId, enum DpTPrPgeStateType& returnState);
```

This operation is mainly used by DpPrDataManager to retrieve the current job state to determine the validity of invocation of DpPrDataManager application. dprId - Data Processing Request Id. returnState - state of the given job is returned.

- GetMachine

```
EcUtStatus GetMachine(const RWCString& dprId, RWCString& returnHost);
```

Returns the updated machine on which the PGE will be running. This operation is mainly used by DpPrDataManager dprId - Data processing request Id. returnHost - Machine returned.

- GetPgeId

```
RWCString GetPgeId();
```

This operation returns the value of the private attribute myPgeId to caller.

- GetRunTimeDir

```
EcUtStatus GetRunTimeDir(const RWCString& dprId, RWCString& returnDir);
```

Retrieves rumtime directory from PCF table and returns to caller. dprId - Data Processing Request Id returnDir - runtime directory is returned to caller.

- GetSswId

```
RWCString GetSswId();
```

Science software ID is returned to caller.

- InsertPgeResourceUsage

```
EcUtStatus InsertPgeResourceUsage(const RWCString& fullLogFile);
```

This private operation performs insertion of resource usage collected after the PGE completed. fullLogFile - file that collects resource usage information of a PGE These collected resource information is populated into DPR_COMPLETION table.

- PostExecute

```
EcUtStatus PostExecute(const RWCString& dprId, EcTInt& pgeExitCode);
```

This operation performs post processing upon the completed run PGE. A report which contains information about the PGE run is generated. Data processing request ID must be supplied. A history log file is created.

- PrepExecute

```
EcUtStatus PrepExecute(const RWCString& dprId, RWCString& returnedCommand, EcTInt debugFlag);
```

This operation performs prep-execution of a PGE run. A Process control file and UNIX profile will be created. dprId - Data processing request Id is required. debugFlag - flag inserted in PCF to run PGE in debug or non-debug mode; 0 = OFF; 1 = ON. pgeCommand - Command to invoke PGS shell is returned. Runtime files used during PGE run will be created.

- ReadFile

```
EcUtStatus ReadFile(const RWCString& fileName, <any> ... , EcTInt& exitCode, RWCString& exitMessage);
```

Retrieves resource usage information from the file created by the PGE wrapper. This private operation is mainly used at Post-execution. fileName - name of the file. fileList - fileName and sizes are returned. exitCode - status of PGE is returned. exitMessage - PGE exit condition message is returned

- RemoveDirectory

```
EcUtStatus RemoveDirectory();
```

This private operation removes directory deallocated by DeallocateDisk. Database record that contains this directory information is deleted.

- RemovePGE

```
EcUtStatus RemovePGE(const RWCString& sswId, const RWCString& machine, EcTInt& deleteFlag);
```

Removes PGE on the given machine if deleteFlag is turned ON. pgeId - Id of PGE software. machine - machine on which the PGE is located.

- RetrievePGEMarkedFiles

```
EcUtStatus RetrievePGEMarkedFiles(const RWCString&  
pcfFullPathName, <any> ...);
```

This private operation reads PCF and retrieves files marked by the PGE during the run. These files will be destaged to Data Server with core (if any) files for debug information.

pcfFullPathName - full path name of PC file fileList - a list of files marked by the PGE is returned.

- RetrieveRuntimeFiles

```
EcUtStatus RetrieveRuntimeFiles(const RWCString& fullPcFileName,  
<any> ...);
```

This private operation collects all runtime file names which include Process Constrol file, profile, SMFs ..., and insert them into fileList. These file will be destaged by DM after PGE run. localPcFile - full path name of PC file on queuing server. A list of runtime file names is returned to caller.

- SendAlarm

```
EcUtStatus SendAlarm(const RWCString& message);
```

Send alarm to Autostys.

- SetJobState

```
EcUtStatus SetJobState(const RWCString& dprId, enum  
DpTPrPgeStateType jobState);
```

This operation sets job state requested by external application. It is mainly used by DpPrDataManager to set state of the DPR after the application completes successfully. dprId - Data Processing Request Id jobState - state of job to be updated in PGE database table.

- SetPGEFailed

```
EcUtStatus SetPGEFailed();
```

Set PGE job in Autostys to failure

- StagePGE

```
EcUtStatus StagePGE(const RWCString& pgeUR, const RWCString&  
topLevelShellName, const RWCString& sswId, EcTFloat  
unTarFileDiskSpace);
```

This private operation performs staging of PGE software on production machine.

topLevelShellName - name of the actual PGE shell script. sswId - science software Id which is

UR for thie PGE package. unTarFileDiskSpace - total disk space of this PGE package after untarred.

- UpdateCommand

```
EcUtStatus UpdateCommand(const RWCString& jobName, const  
RWCString& command);
```

This private operation performs update command in Autosys database with new command information which is used to invoke the PGE wrapper on production machine. This method is normally used at the end of Pre-execution. jobName - name of job defined in Autosys database for the actual PGE run command - new command information to invoke the PGE wrapper.

- UpdateDprStatus

```
EcUtStatus UpdateDprStatus();
```

Updates the status of DPR in DPR_COMPLETION table (?).

- UpdateMachine

```
EcUtStatus UpdateMachine(const RWCString& jobName, const  
RWCString& machine);
```

This operation performs update machine in Autosys database with new machine returned from Resource Manager. jobName - name of job defined in AutoSys database machine - new machine on which this jobName to be running

7.3.4.3.2 DpPrLock Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

```
myFd: EcTInt
```

Constructors and Destructor:

```
DpPrLock(const EcTChar* criticalPath);
```

```
~DpPrLock();
```

Operations:

7.3.4.3.3 DpPrPcf Class

Overview:

Export Control: **Public**

Inheritance Relationships:

Attributes:

myDprId: **RWCString**

myFileInfoList: **RWTValslist**

myFlag: **EctInt**

Debug flag, used to create PCF in debug mode.

myHost: **RWCString**

myLocation: **RWCString**

myName: **RWCString**

myPermission: **RWCString**

myPgeId: **RWCString**

mySswId: **RWCString**

```
myStatus: EcUtStatus
```

```
myType: enum DpTPrPcfType
```

Constructors and Destructor:

```
DpPrPcf(const RWCString& dprId, enum DpTPrPcfType fileType);
```

This operation construct a DpPrPcf object with dprId and fileType as two keys to retrieve the actual DpPrPcf object from PCF table. Use the Read operation to retrieve the object from PCF table. dprId: Job Id that uses this file at runtime. fileType: type of file (PCF or UNIX profile) A DpPrPcf object is constructed.

```
DpPrPcf(EcUtStatus& returnStatus, const RWCString& dprId, const RWCString& name, const RWCString& location, const RWCString& access, enum DpTPrPcfType fileType);
```

This operation creates a physical file of type specified by fileType. Each creation of the file causes a row inserted into the PCF table in database. All the neccessary information to create the file must be supplied. The information includes dprId, pgeId, file name, file location, file permission and file type. A file of type fileType is created at location specified. A record is inserted into table PCF. Use Insert to insert The object to database table.

```
DpPrPcf();
```

This default constructor instantiate a Pcf object so that this object can create Process control file, Profile, and Processing history file for each PGE run. An object of type DpPrPcf is created.

```
virtual ~DpPrPcf();
```

This destructor will perform an orderly cleanup and deletion of the object. The object of type DpPrPcf is orderly destroyed.

Operations:

- CreateFile

```
EcUtStatus CreateFile();
```

This operation creates the file based on data given at the alternate constructor. A file is created. A record is inserted in to PCF table.

- CreatePcf

```
EcUtStatus CreatePcf();
```

This operation creates the physical Process Control file at local disk A file is created at local disk

- CreateProfile

```
EcUtStatus CreateProfile();
```

This operation create a UNIX profile for a PGE run. A UNIX profile is created at local disk.

- Exist

```
RwCString GetDprId();
```

Check for the existence of PCF record. Boolean status is returned in output parameter. EcDTrue: record exists. EcDFalse: record does not exist.

- GetDprId

```
RwCString GetDprId();
```

The value of myDprId is returned to caller.

- GetFileInfoList

```
RWTValslist GetFileInfoList();
```

This operation return a list of watched files information to caller.

- GetHost

```
RwCString GetHost();
```

The value of the private attribute is returned. None.

- GetInputInfo

```
EcUtStatus GetInputInfo(const RwCString& ur, const RwCString& machine, RwCString& returnFileName, RwCString& returnPath);
```

Interfaces with Data Manager to retrieve name and location of input granules. ur - UR retrieved from PLANG. machine - Machine on which the PGE is running. returnFileName - Actual name of the file is returned. returnPath - Location of this file name is returned.

- GetLevelZeroAttributes

```
EcUtStatus GetLevelZeroAttributes(const RwCString& urId, RwCString& returnedStartOrbitNum, RwCString& returnedEndOrbitNum, RwCString& returnedAPID);
```

Inspects the given UR, and retrieves attributes needed in the PCF for Level zero data granules. urId - UR which define level zero data. returnedStartOrbitNum - start orbit number is returned. returnedStopOrbitNum - stop orbit number is returned.

- GetLocation

```
RwCString GetLocation();
```

Retrieve the directory path of the Process Control File (PCF), as defined by the Location attribute, and return it to the calling process.

- GetMasterProfile

```
EcUtStatus GetMasterProfile();
```

This operations copy profile from PGE location to runtime directory. Status of the process is returned.

- GetName

```
RWCString GetName();
```

Retrieve the filename of the Process Control File (PCF), as defined by the Name attribute, and return it to the calling process.

- GetOutputInfo

```
EcUtStatus GetOutputInfo(const RWCString& fileName, const  
RWCString& machine, RWCString& returnPath, EcTFloat&  
returnSize);
```

Interfaces with DpPrDataManager and DpPrDataInfo classes to retrieve path and size for output granules. fileName - name of file in input granule list. machine - machine on which the PGE is running. returnPath - location of output files. returnSize - estimated size for ouput files.

- GetPermission

```
RWCString GetPermission();
```

Retrieve the system permission settings for the Process Control File (PCF), as defined by the Permission attribute, and return it to the calling process.

- GetType

```
enum DpTPrPcfType GetType();
```

This operation returns the current value of myType to the calling process.

- InOutGenerator

```
EcUtStatus InOutGenerator(PlTGranIO ioFlag, <any> ..., PlDpr&  
dprObject, <any> ...);
```

This operation generates version number for input/output files retrieved from PLANG granule list.

- Insert

```
EcUtStatus Insert();
```

This operation inserts DpPrPcf object into PCF table. A record is inserted into PCF table.

- MergeFiles

```
EcUtStatus MergeFiles(RWSlistCollectables&  
generatedSectionList, RWSlistCollectables& sectionList);
```

Merge template PCF and new PCF created. generatedSectionList: PCF generated with PLANG info. sectionList: template PCF. sectionList: this list will be modified.

- Modify

```
EcUtStatus Modify(const RWCString& genString, <any> ...,  
EcTBoolean& exists);
```

If genString exists in section list, then modify section list with this genString. genString: generated string to be written into PCF. section: Process Control file. exists: existence of genString in section.

- Read

```
EcUtStatus Read();
```

This operation retrieves DpPrPcf object from PCF table. An object from PCF table is retrieved.

- Remove

```
EcUtStatus Remove();
```

This operation remove the corresponding file from local disk, and deletes its record in the database. The file is removed from local disk; space is deallocated. A record which monitors this file is deleted.

- RWDECLARE_COLLECTABLE

```
int RWDECLARE_COLLECTABLE(DpPrPcf );
```

- SetFileType

```
EcTVoid SetFileType(enum DpTPrPcfType fileType);
```

Set the type of file DpEPrPCF: Process Control file. DpEPrPROFILE: UNIX profile. DpEPrHISTORY: processing history file. Appropriate file type should be given as input parameter.

- SetFlag

```
EcTVoid SetFlag(EcTInt flagValue);
```

Set debug flag so that PCF can be created with debug info. Flag value should be given as input parameter.

- SetNewLocation

```
EcUtStatus SetNewLocation(const RWCString& newLocation);
```

To cover the possibility that the initially allocated disk resources need to be reallocated, this operation provides for the modification of the Location attribute. A new location must be supplied. The file must be at the new location with information in the database table updated.

7.3.4.3.4 DpPrPge Class

Overview:

Export Control: **Public**

Inheritance Relationships:

Attributes:

myCommands: RWCString

The command string will be used to provide startup condition information to the activation shell identified by the Shell attribute. The default command string is specific to the default SDP Toolkit activation shell.

myCpuResource: EcTUShortInt

This attribute determines how much processing resource needed for this Pge run.

myDprId: RWCString

This attribute together with mySswId attribute will uniquely determine a Pge object.

myEnvironment: RWCString

This attribute may contain the value of zero or more environment variable pairs which will be used to define the operating conditions for the science software.

myExecName: RWCString

This attribute defines the name of actual PGE script.

myFileInfoList: RWTValslist

This attribute defines a list of output file names to be watched during the PGE run.

myFlag: EcTInt

ECS debug attribute

myHost: RWCString

This attribute identifies the host machine for this instance of the Pge object. There will be a most one instance of this object per host.

myPgeId: RWCString

This attribute is used as pgeId in Process Control file.

myShell: RWCString

This attribute defines the Processing shell which activates the science software's outer PGE shell. The default shell is provided by the SDP Toolkit.

mySswId: RWCString

This attribute is used to uniquely identify the science software which is occupying a particular machine. The same identifier may be used as the value of the PgeId attribute for another instance of this object provided that the value of the Host attribute is different.

myState: enum DpTPrPgeStateType

The state attribute maintains the last known state of the PGE object. Internal activation of the CheckStatus operation will update this value on a periodic basis.

Constructors and Destructor:

DpPrPge(const RWCString& dprId);

This constructor is used to retrieve a DpPrPge object from database. Use Read operation to retrieve the object. Data processing request id must be supplied. myDprId will initialized with the given dprId.

DpPrPge(EcUtStatus& returnStatus, const RWCString& dprId, const RWCString& sswId, const RWCString& pgeHost, EcTUShortInt cpu, const RWCString& execName, enum DpTPrPgeStateType pgeState);

This constructor will be used to create a PGE object to monitor a Pge run. Use Insert to insert this DpPrPge object into PGE table.

DpPrPge();

This constructor will be used to create a PGE object to monitor a Pge run. None. A default constructor is created.

virtual ~DpPrPge();

This operation orderly destroys DpPrPge object. Memory is deallocated.

Operations:

- CreateRunTimeFile

EcUtStatus CreateRunTimeFile(RWCString& returnedFullPathName, enum DpTPrPcfType fileType);

This private operation creates runtime files for a run of a PGE. Type of the file to be created is required. A memory address space is provided for full path name of the created file. A full path name of the created file is assigned to the provided memory address space.

- Delete

EcUtStatus Delete();

Delete a PGE record based on DprId. This method is normally invoked during DeallocationResource method.

- Execute

```
EcUtStatus Execute(const RWCString& pgeCommands, const  
RWCString& pgeEnvironment, EcTInt debugFlag, const RWCString&  
pgeShell);
```

This operation will prepare the execution for the actual execution invoked by autoSys.
pgeCommands: Command set for Pge script. pgeEnvironment: Environment variable contains location of the PGE. pgeShell: The Shell which invoke the actual PGE script. Runtime files will be created State of the Pge will be updated in PGE table.

- GetCommands

```
RWCString GetCommands();
```

Retrieves the contents of the Commands attribute and returns it to the calling process.

- GetCpuResource

```
EcTUShortInt GetCpuResource();
```

This operation returns the private attribute myCpuResource to provide ResourceManager how much CPU is needed for a PGE run

- GetDprId

```
RWCString GetDprId();
```

Retrieves the value of the PgeId attribute and returns it to the calling process.

- GetEnv

```
RWCString GetEnv();
```

Retrieves the contents of the Environment attribute and returns it to the calling process.

- GetExecName

```
RWCString GetExecName();
```

Name of the actual PGE script is returned. The value of the private attribute myExecName is returned.

- GetFileInfoList

```
RWTValslist GetFileInfoList();
```

File information created when PCF is created is returned.

- GetFlag

```
EcTInt GetFlag();
```

Return the value of the private attribute myFlag.

- GetHost

```
RWCString GetHost();
```

Retrieves the value of the Host attribute and returns it to the calling process.

- GetPID

```
RWCString GetPID();
```

Temporarily uses myEnvironment as PID.

- GetShell

```
RWCString GetShell();
```

Retrieves the name of the activation shell, as defined by the Shell attribute, which will be used by the Processing System to cradle the science software.

- GetSswId

```
RWCString GetSswId();
```

Retrieves the value of the PgeId attribute and returns it to the calling process.

- GetState

```
enum DpTPrPgeStateType GetState();
```

Retrieves the state of the object as currently defined by the State attribute.

- Insert

```
EcUtStatus Insert();
```

This operation is used to check for valid PGE record first then inserts this DpPrPge object into PGE table. Data member initialization has been done in the constructor. A record is created in PGE table.

- Read

```
EcUtStatus Read();
```

This operation retrieves DpPrPge object from database into memory. A new DpPrPge object is created with information from PGE table.

- RWDECLARE_COLLECTABLE

```
int RWDECLARE_COLLECTABLE(DpPrPge );
```

- SetPID

```
EcTVoid SetPID(const RWCString& spid);
```

- SetState

```
EcTVoid SetState(enum DpTPrPgeStateType state);
```

- Stage

```
EcUtStatus Stage(const RWCString& pgeUR, const RWCString& sswId,
    const RWCString& topLevelShellName, const RWCString& hostName,
    enum DpTPrComponentType elementType, const RWCString& basePath);
```

This operation will be used to effect the transfer of science software executables and Status Message Files (SMFs). Use of this operation requires that all of the required disk resources be allocated ahead of time.

- Update

```
EcUtStatus Update();
```

7.3.4.3.5 DpPrUtil Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

myGenProcPtr: EcPfGenProcess*

Global GenProcess pointer

results: RWCString&

Constructors and Destructor:

DpPrUtil();

The default constructor, destructor of the interface manager is protected as per singleton design pattern

```
~DpPrUtil();
```

Operations:

- Auth

```
static EcUtStatus Auth(RWCString& userName, RWCString&
userPassword);
```

NAME DpPrUtil::Auth DESCRIPTION Retrieves database login from Security class, and reads the Configuration file that holds the values of database information. A Configuration file must be previously registered.

- DbConnect

```
static EcUtStatus DbConnect();
```

NAME DpPrUtil::DbConnect DESCRIPTION Performs database connections for db I/F object for DpPrPge, DpPrPcf, DpPrExecutable classes. All database interface objects is connection to server.

- DirectoryEntries

```
static EcTInt DirectoryEntries(EcUtStatus& status, const
RWCString& pathName);
```

XNCR 17 Replaced by ECSed05271

NAME DpPrUtil::DirectoryEntries DESCRIPTION This operation performs a check on the directory located at 'pathName' to determine if it is currently empty. The actual number of entries gets returned if the call was successful.

- DpPrGenerateMetFileFromTemplate

```
static EcUtStatus DpPrGenerateMetFileFromTemplate(RWCString&
templateName, RWCString& newFileName, <any> ...);
```

NAME DpPrGenerateMetFileFromTemplate DESCRIPTION This operation generates a metadata file from a template containing placeholders of the form '\$N', where N is any integer greater than zero. These placeholders are replaced by the corresponding parameter in the list. For example:

VALUE = \$3

becomes

VALUE = 1998-01-01T12:00:00

if *parameters[2] = "1998-01-01T12:00:0"

The placeholders do not have to occur in order (\$1, \$2, \$3...\$N) The placeholder must be at the end of the line. A parameter must exist for each placeholder in the template, even if it is "".

templateName - the full pathname of the metadata template newFileName - the full pathname of the new metadata file parameters - an ordered vector of parameters to put in the new metadata file (element 0 corresponds to parameter #1 in the template file)

- DpPrGenerateODLFileFromTemplate

```
static EcUtStatus DpPrGenerateODLFileFromTemplate(RWCString&
templateName, RWCString& newFileName, <any> ..., <any> ...);
```

FIX XNCR 27 Production History

Replaced by ECSed?????

NAME DpPrGenerateODLFileFromTemplate DESCRIPTION This operation generates a Production History file from an ODL template containing placeholders of the form '\$N', where N is any integer greater than zero. These placeholders are replaced by the corresponding parameter in the user supplied list, according to the groupings defined in a second list which determines the starting location for parameter replacement For example:

GROUP = USERINFO OBJECT = NAME VALUE = \$1 OBJECT = PHONE VALUE = \$2
OBJECT = ADDRESS VALUE = \$3

GROUP = DATACOLLECTION OBJECT = LOCATION VALUE = \$1 OBJECT = ORBIT
VALUE = \$2 OBJECT = DATETIME VALUE = \$3 becomes VALUE = 1998-01-01T12:00:00
if *groups[2] = "DATACOLLECTION", *groups[3]=3

AND

if *parameters[2] = "1998-01-01T12:00:0"

n.b. *groups[0] = "USERINFO" and *groups[1] = 0

The placeholders do not have to occur in order (\$1, \$2, \$3...\$N) within the template file. However, each placeholder must appear at the end of the line. Additionally, a parameter must exist for each placeholder in the template, even if it is "". The addition of a grouping list, with parameter offsets, allows for replication of grouping information, without requiring the ODL template to define the number of such groupings. For example:

GROUP = USERINFO OBJECT = NAME VALUE = \$1 OBJECT = PHONE VALUE = \$2
OBJECT = ADDRESS VALUE = \$3

can be replicated many times:

if *groups[0] = "USERINFO" and *groups[1] = 0 then *parameters[0] = "DAVID"
*parameters[3] = "JOE" *parameters[6] = "SONG"

- DsAcquire

```
static EcUtStatus DsAcquire(const RWCString& targetHost, const
RWCString& destinationPath, const RWCString& urName);
```

- DsInspect

```
static EcUtStatus DsInspect(const RWCString& userName, const  
RWCString& urId, RWCString& inBeginOrbit, RWCString&  
outEndOrbit, RWCString& apId);
```

NAME DpPrUtil::DsInspect DESCRIPTION Performs inspection of URs. userName - user name. urId - UR to be inspected.

- ExecuteCommand

```
static EcUtStatus ExecuteCommand(const RWCString& unixCommand);
```

NAME DpPrUtil::ExecuteCommand DESCRIPTION This method performs execution of UNIX command. UNIX command string.

- GetConfigValue

```
static EcUtStatus GetConfigValue(const RWCString& name,  
RWCString& value);
```

NAME GetConfigValue DESCRIPTION This method provides interface to retrieve information from configuration file. name - key defined in config file. value - output parameter retrieved from config file.

- GetEnvDbValues

```
static EcUtStatus GetEnvDbValues(RWCString& dbLoginName,  
RWCString& dbLoginPassword, RWCString& dbName, RWCString&  
dbServer, RWCString& dbLib);
```

NAME DpPrUtil::GetEnvDbValues DESCRIPTION Retrieves database login information from configuration file.

- InitErrorList

```
static EcTVoid InitErrorList();
```

NAME DpPrUtil::InitErrorList DESCRIPTION Performs initialization of error list. This list is used to interpret error code, returned from EM software, to string which has been stored corresponding to the error code. This string will be sent to AutoSys as an alarm message.

- InterruptExist

```
static EcTBoolean InterruptExist(const RWCString& methodName,  
EcUtStatus& thisStatus);
```

NAME InterruptExist DESCRIPTION Check for the existence of interrupt signal. Method name where interrupt signal occurs Status is returned as output parameter.

- NameExist

```
static EcUtStatus NameExist(const RWCString& pathName,  
EcTBoolean& exist, enum DpTPrElementType& nameType);
```

NAME DpPrUtil::NameExist
DESCRIPTION Checks the existence of given file name. exist - EcDTrue if name exists. EcDFalse otherwise. pathName - Absolute path/name. nameType - Output parameter to indicate whether name is file (DpEPrEMFile) or directory (DpEPrEMDirectory). Status of the method is returned.

- Remove

```
static EcUtStatus Remove(const RWCString& pathName);
```

NAME DpPrUtil::Remove
DESCRIPTION This operation performs removal of file name at local machine pathName - full file name to be removed. The given file is removed at local machine.

- ResultsCommand

```
static EcUtStatus ResultsCommand(const RWCString& unixCommand);
```

NAME DpPrUtil::ResultsCommand
DESCRIPTION This method performs execution of UNIX command, but returns the results of running the command.

- StatusCommand

```
static EcUtStatus StatusCommand(const RWCString& unixCommand,  
EcTInt& unixStatus);
```

NAME DpPrUtil::StatusCommand
DESCRIPTION This method performs execution of UNIX command, but returns the status of the command itself.

- TarRuntimeFiles

```
static EcUtStatus TarRuntimeFiles(const RWCString&  
tarFullPathName, const RWTValSlist& fileList);
```

Tar all file names stored in fileList. The tar file is named as given tarFullPathName.

7.3.4.3.6 PICompletion Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

myCompletionDate: PlTime

The Completion date is the date the DPR completed This information is not updated by this class; it is nonetheless included here for completeness.

`myDprElapsedTIme: EcTFloat`

The wall-clock time that elapsed to complete the entire DPR process.

`myDprId: RWCString`

The DPR identifier for which resource usage statistics are being gathered.

`myExitCode: EcTInt`

The exit condition code returned by the PGE.

`myPgeBlockInputs: EcTInt`

The number of Unix service requests required to sustain read operations on performed by the PGE.

`myPgeBlockOutputs: EcTInt`

The number of Unix service requests required to sustain write operations on performed by the PGE.

`myPgeCpuTime: EcTFloat`

CPU execution time incurred by the User.

`myPgeDynamicMemory: EcTFloat`

The maximum resident memory set size in pages.

`myPgeElapsedTIme: EcTFloat`

The wall-clock time that elapsed to complete just the PGE component of the DPR.

`myPgeFaults: EcTInt`

The number of times the PGE incurred a page fault which resulted in some actual disk I/O.

`myPgeSharedMemory: EcTFloat`

Currently Unused

`myPgeSwaps: EcTInt`

The number of times a part of the PGE was swapped out of main memory.

`myPgeSysTime: EcTFloat`

CPU execution time incurred by the System.

`myPlatform: RWCString`

The science processing machine used by this PGE/DPR.

Constructors and Destructor:

```
PlCompletion(EcUtStatus& status, const RWCString& dprId);
```

Construct an object for recording the PGE/DPR resource usage statistics.

```
~PlCompletion();
```

Remove the object from memory

Operations:

- DbConnect

```
EcUtStatus DbConnect();
```

- Read

```
EcUtStatus Read(const RWCString& dprId, const RWCString& platform);
```

- SetDprDate

```
EcTVoid SetDprDate(PlTime date);
```

Record the completion date of the DPR.

- SetDprElapsed

```
EcTVoid SetDprElapsed(EcTFloat deltaTime);
```

Record the total elapsed time that it took for the DPR to complete.

- SetPgeBlockInput

```
EcTVoid SetPgeBlockInput(EcTInt blockInput);
```

Record the total number of block input operations that occurred during the run of the PGE.

- SetPgeBlockOutput

```
EcTVoid SetPgeBlockOutput(EcTInt blockOutput);
```

Record the total number of block output operations that occurred during the run of the PGE.

- SetPgeDynMem

```
EcTVoid SetPgeDynMem(EcTFloat maxMem);
```

Record the resident set size of the PGE.

- SetPgeElapsed

```
EcTVoid SetPgeElapsed(EcTFloat elapsedTime);
```

Record the total elapsed time that it took for the PGE to complete.

- SetPgeExitCode

```
EcTVoid SetPgeExitCode(EcTInt exitCode);
```

Report the final exit condition which was returned by the PGE to the processing system.

- SetPgeFaults

```
EcTVoid SetPgeFaults(EcTFloat pageFaults);
```

Report on the number of major page faults that occurred during execution of the PGE.

- SetPgePageSwaps

```
EcTVoid SetPgePageSwaps(EcTInt pageSwaps);
```

Record the total number of page swaps that occurred during the run of the PGE.

- SetPgePlatform

```
EcTVoid SetPgePlatform(RWCString host);
```

Report the actual science processing hardware that was used for this execution of the PGE/DPR

- SetPgeSharedMem

```
EcTVoid SetPgeSharedMem(EcTFloat sharedMem);
```

Record the amount of shared memory consumed by the PGE.

- SetSysTime

```
EcTVoid SetSysTime(EcTFloat sysTime);
```

Report on the amount of System time spent during execution.

- SetUsrTime

```
EcTVoid SetUsrTime(EcTFloat usrTime);
```

Report on the amount of User time spent during execution.

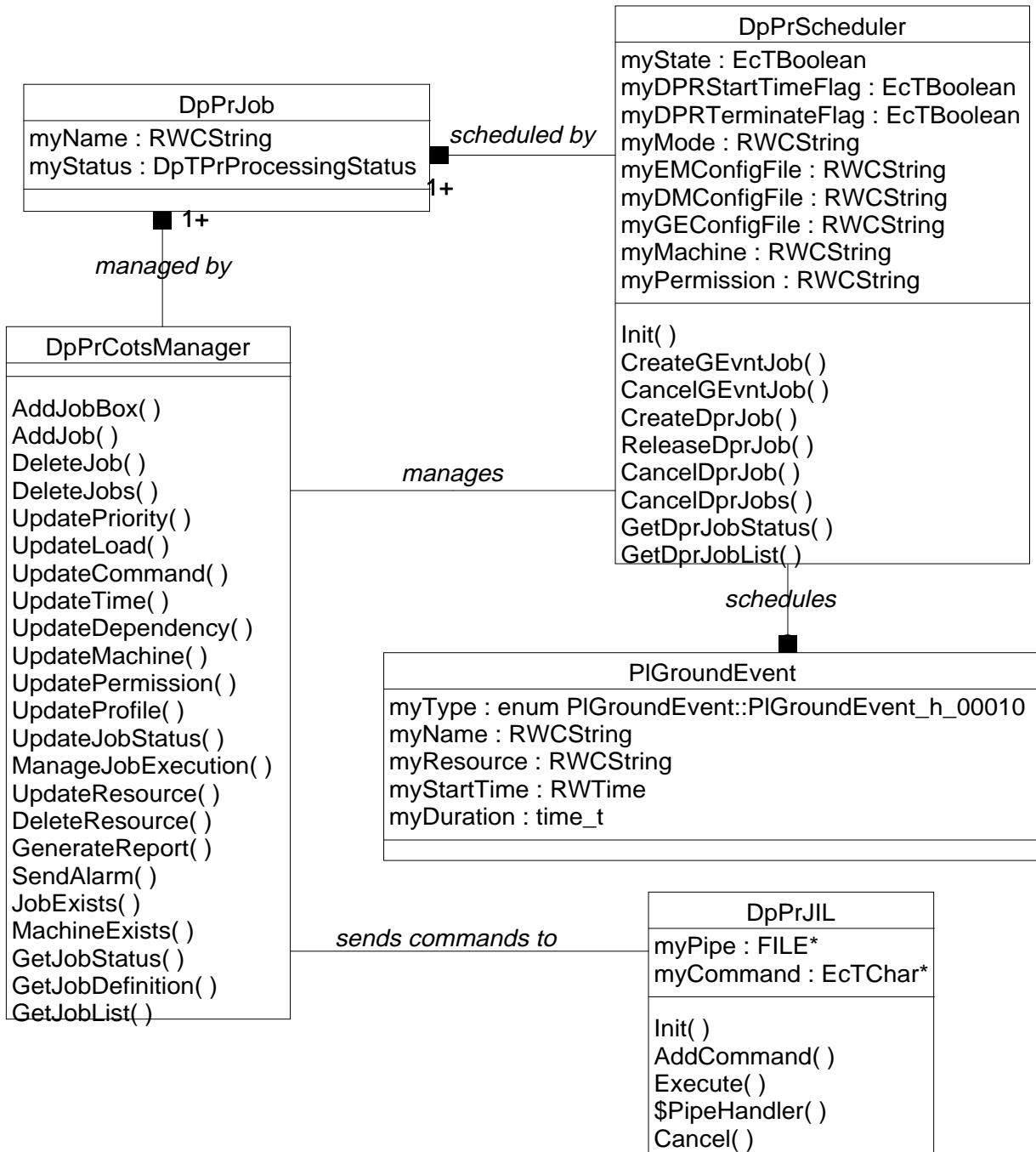
- Write

```
EcUtStatus Write();
```

Load the new object contents into the PDPS database

7.3.5 Processing_Job_Management Class Category

7.3.5.1 Overview



7.3.5.2 Processing_Job_Management Classes

7.3.5.3 DpPrCotsManager Class

Overview:

This class provides an interface to the AutoSys COTS product to add, delete, and modify jobs and machines; control job processing; retrieve job definition and processing status; generate reports; and send alarms.

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

`DpPrCotsManager();`

This is the constructor.

`virtual ~DpPrCotsManager();`

This is the destructor.

Operations:

- AddJobBox

```
EcUtStatus AddJobBox(const RWCString& name, const RWCString&
tobox, const RWCString& daysofweek, const RWCString& starttime,
const RWCString& permission, EcTInt jobload, DpTPrJobPriority
priority, const RWTValslist& dependlist);
```

This public member function inserts a job box.

- AddJob

```
EcUtStatus AddJob(const RWCString& name, const RWCString& tobox,
const RWCString& daysofweek, const RWCString& starttime, const
RWCString& commandline, const RWCString& machine, const
RWCString& permission, const RWCString& profile, EcTInt
avgruntime, EcTInt maxruntime, EcTBoolean hold, EcTInt jobload,
DpTPrJobPriority priority, EcTInt maxexitcode, const
RWTValslist& dependlist);
```

This public member function inserts a job.

- DeleteJobs

```
EcUtStatus DeleteJobs(const RWTValSlist& joblist);
```

This public member function deletes job(s).

- DeleteJob

```
EcUtStatus DeleteJob(const RWCString& job);
```

This public member function deletes a job.

- DeleteResource

```
EcUtStatus DeleteResource(const RWCString& machine);
```

This public member function deletes a machine resource.

- GenerateReport

```
EcUtStatus GenerateReport(const RWCString& name, DpTPrReportType type, EcTBoolean detail, RWCString& report);
```

This public member function generates a report.

- GetJobDefinition

```
EcUtStatus GetJobDefinition(const RWCString& job, RWCString& JILdefinition);
```

This public member function returns a job's definition in AutoSys Job Information Language (JIL) format.

- GetJobList

```
EcUtStatus GetJobList(DpTPrProcessingStatus jobstatus, EcTBoolean boxonly, <any> ...);
```

This public member function returns a list of job(s).

- GetJobStatus

```
EcUtStatus GetJobStatus(const RWCString& job, DpTPrProcessingStatus& jobstatus);
```

This public member function returns a job's processing status.

- JobExists

```
EcUtStatus JobExists(const RWCString& job);
```

This private helper function verifies the existence of a job.

- MachineExists

```
EcUtStatus MachineExists(const RWCString& machine);
```

This private helper function verifies the existence of a machine resource.

- ManageJobExecution

```
EcUtStatus ManageJobExecution(const RWCString& job,  
DpTPrExecution flag);
```

This public member function provides an interface to manually control job execution.

- SendAlarm

```
EcUtStatus SendAlarm(DpTPrAlarmType alarm, const RWCString&  
comment);
```

This public member function sends an alarm.

- UpdateCommand

```
EcUtStatus UpdateCommand(const RWCString& job, const RWCString&  
commandline);
```

This public member function updates a job's command attribute.

- UpdateDependency

```
EcUtStatus UpdateDependency(const RWCString& job, const  
RWTValslist& dependlist);
```

This public member function updates a job's starting condition dependency attribute.

- UpdateJobStatus

```
EcUtStatus UpdateJobStatus(const RWCString& job,  
DpTPrProcessingStatus jobstatus);
```

This public member function updates a job's processing status.

- UpdateLoad

```
EcUtStatus UpdateLoad(const RWCString& job, EcTInt jobload);
```

This public member function updates a job's required load attribute.

- UpdateMachine

```
EcUtStatus UpdateMachine(const RWCString& job, const RWCString&  
machine);
```

This public member function updates a job's machine attribute.

- UpdatePermission

```
EcUtStatus UpdatePermission(const RWCString& job, const  
RWCString& permission);
```

This public member function updates a job's permission.

- UpdatePriority

```
EcUtStatus UpdatePriority(const RWCString& job, DpTPrJobPriority priority);
```

This public member function updates a job's priority attribute.

- UpdateProfile

```
EcUtStatus UpdateProfile(const RWCString& job, const RWCString& profile);
```

This public member function updates a job's Bourne shell profile attribute.

- UpdateResource

```
EcUtStatus UpdateResource(const RWCString& machine, EcTInt maxload, EcTFloat factor);
```

This public member function updates a machine resource.

- UpdateTime

```
EcUtStatus UpdateTime(const RWCString& job, const RWCString& daysofweek, const RWCString& starttime, EcTInt avgruntime, EcTInt maxruntime);
```

This public member function updates a job's time constraints attribute.

7.3.5.3.1 DpPrJIL Class

Overview:

This class provides an Application Program Interface (API) to the AutoSys Job Information Language (JIL) processor.

Export Control: [Public](#)

Inheritance Relationships:

Attributes:

myCommand: EcTChar*

This private data member contains the cached command(s).

myPipe: FILE*

This private data member contains the pipe handle.

Constructors and Destructor:

```
DpPrJIL();
```

This is the constructor.

```
virtual ~DpPrJIL();
```

This is the destructor.

Operations:

- AddCommand

```
EcUtStatus AddCommand(const EcTChar* command);
```

This public member function adds the user specified command to the internal command buffer.

- Cancel

```
EcTVoid Cancel();
```

This public member function terminates the AutoSys JIL process without executing any cached command(s).

- Execute

```
EcUtStatus Execute();
```

This public member function executes the previously cached command(s) and terminates the AutoSys JIL process.

- Init

```
EcUtStatus Init();
```

This public member function creates an AutoSys JIL process.

- PipeHandler

```
static EcTVoid PipeHandler();
```

This private static function is the pipe signal handler.

7.3.5.3.2 DpPrJob Class

Overview:

This container class holds the Data Processing Request's (DPR) processing status information.

Export Control: Public

Inheritance Relationships:

Attributes:

myName : RWCString

This public data member contains the DPR name.

myStatus : DpTPrProcessingStatus

This public data member contains the DPR processing status.

Constructors and Destructor:

DpPrJob();

This is the constructor.

DpPrJob(const RWCString& name, DpTPrProcessingStatus status);

This is an alternate constructor that sets the DPR name and processing status.

virtual ~DpPrJob();

This is the destructor.

Operations:

7.3.5.3.3 DpPrScheduler Class

Overview:

This class provides PLANG Planning Workbench and Subscription Manager components a programmatic interface to the PRONG. Operations to create and cancel Data Processing Requests (DPR) and Ground Events are provided. Also provided are methods to release a DPR for execution, retrieve the processing status of a DPR, and query for a list of DPRs currently in the schedule.

Export Control: Public

Inheritance Relationships:

Attributes:

myDMConfigFile : RWCString

This private data member contains the Process Framework configuration file specification for Data Manager application.

myDPRStartTimeFlag : EcTBoolean

This private data member contains the Start Time assignment flag. If set to true, Start Time attribute is assigned when a DPR job is created.

myDPRTerminateFlag: EcTBoolean

This private data member contains the DPR terminate flag. If set to true, the DPR box job will terminate if any administrative jobs in it fails.

myEMConfigFile: RWCString

This private data member contains the Process Framework configuration file specification for Execution Manager application.

myGEConfigFile: RWCString

This private data member contains the Process Framework configuration file specification for Ground Event application.

myMachine: RWCString

This private data member contains the machine specification.

myMode: RWCString

This private data member contains the Process Framework mode specification.

myPermission: RWCString

This private data member contains the permission specification.

myState: EcTBoolean

This private data member contains the state of the object.

Constructors and Destructor:

DpPrScheduler();

This is the constructor.

virtual ~DpPrScheduler();

This is the destructor.

Operations:

- CancelDprJobs

```
EcUtStatus CancelDprJobs(enum  
DpPrScheduler::DpPrScheduler_h_00010 type);
```

This public member function cancels DPR job(s). Inactive DPR jobs are deleted, and active DPR jobs are terminated.

- CancelDprJob

```
EcUtStatus CancelDprJob(PlDpr& dpr);
```

This public member function cancels a DPR job. If the DPR job is currently running, it is terminated. Otherwise, the DPR job is deleted.

- CancelGEvntJob

```
EcUtStatus CancelGEvntJob(PlGroundEvent& event);
```

This public member function cancels a Ground Event job. If the Ground Event job is currently running, it is terminated. Otherwise, the Ground Event job is deleted.

- CreateDprJob

```
EcUtStatus CreateDprJob(PlDpr& dpr, const RWTValslist&  
dprdependlist, EctBoolean hold);
```

This public member function creates a DPR job and starts the job, if necessary.

- CreateGEvntJob

```
EcUtStatus CreateGEvntJob(PlGroundEvent& event);
```

This public member function creates a Ground Event job and starts the job, if necessary.

- GetDprJobList

```
EcUtStatus GetDprJobList(DpTPrProcessingStatus dprstatus, <any>  
...);
```

This public member function returns a list of DPR job(s).

- GetDprJobStatus

```
EcUtStatus GetDprJobStatus(PlDpr& dpr, DpTPrProcessingStatus&  
dprstatus);
```

This public member function returns a DPR job's processing status.

- Init

```
EcUtStatus Init();
```

This public member function performs Process Framework initialization.

- ReleaseDprJob

```
EcUtStatus ReleaseDprJob(PlDpr& dpr);
```

This public member function releases a DPR job.

7.3.5.3.4 **PIGroundEvent Class**

Overview:

This container class holds the Ground Event information.

Export Control: **Public**

Inheritance Relationships:

Attributes:

myDuration: time_t

This public data member contains the Ground Event duration attribute.

myName: RWCString

This public data member contains the Ground Event name.

myResource: RWCString

This public data member contains the Ground Event computer resource name.

myStartTime: RWTime

This public data member contains the Ground Event start time attribute.

myType: enum PlGroundEvent::PlGroundEvent_h_00010

This public data member contains the Ground Event type.

Constructors and Destructor:

PlGroundEvent();

This is the constructor.

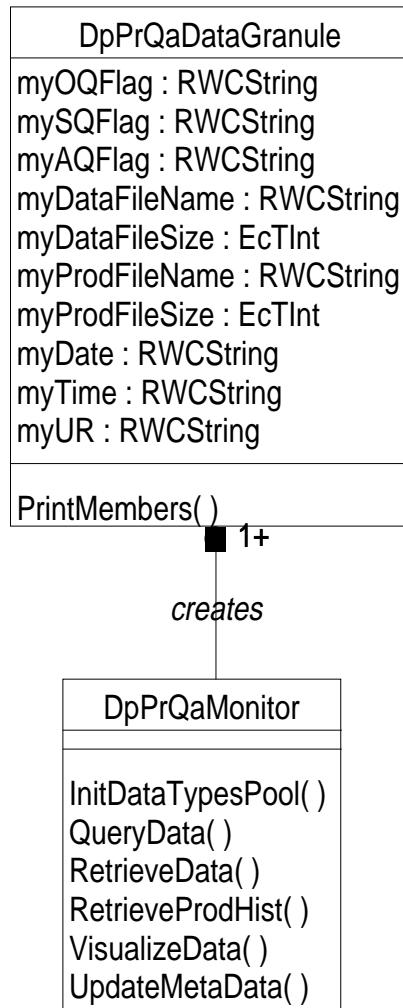
virtual ~PlGroundEvent();

This is the destructor.

Operations:

7.3.6 Processing_QA_Monitor Class Category

7.3.6.1 Overview



7.3.6.2 Processing_QA_Monitor

7.3.6.3 DpPrQaDataGranule

Overview:

It defines attributes and operations of data parameters returned from DataServer when operator tries to query data for a particular data type during a period of starting and ending day

The data members of this class are columns that are display in data granules list in QaMonitor GUI.

Export Control: Public

Inheritance Relationships:

Attributes:

myAQFlag: RWCString

Automatic Quality Flag

myDataFileName: RWCString

This is FileName of a particular data granule

myDataFileSize: EcTInt

File size of a particular data granule

myDate: RWCString

date of data granule insertion

myOQFlag: RWCString

Operational Quality Flag

myProdFileName: RWCString

This is FileName of a particular production history

myProdFileSize: EcTInt

File size of a particular production history

mySQFlag: RWCString

Science Quality Flag

myTime: RWCString

time of data granule insertion

myUR: RWCString

This is UR of a particular data granule

Constructors and Destructor:

```
DpPrQaDataGranule(const DpPrQaDataGranule& dataGranObj);
```

Copy constructor, the assignment operator and the equality operator are provided so that objects of this class can be stored in RWTValSlist A public copy constructor operation is provided in order to insert this object info object list of type RWTValSlist<T>.

```
DpPrQaDataGranule(RWCString oqFlag, RWCString sqFlag, RWCString  
aqFlag, RWCString fileName, EcTInt fileSize, RWCString urId);
```

The alternate constructor for this class.

```
DpPrQaDataGranule();
```

The default constructor for this class which initializes myFilesize to 0.

```
~DpPrQaDataGranule();
```

The default destructor for this class.

Operations:

- GetAQFlag

```
RWCString GetAQFlag() const;
```

A public operation for other class(es) to use to get the myAQFlag attribute value.

- GetDataFileName

```
const RWCString GetDataFileName();
```

A public operation for other class(es) to use to get myDataFileName attribute value for data granule.

- GetDataFileSize

```
EcTInt GetDataFileSize();
```

A public operation for other class(es) to use to get myDataFileSize attribute value data granule.

- GetDate

```
RWCString GetDate() const;
```

A public operation for other class(es) to use to get myDate attribute value.

- GetOQFlag

```
RWCString GetOQFlag() const;
```

A public operation for other class(es) to use to get the myOQFlag attribute value.

- GetProdFileName

```
const RWCString GetProdFileName();
```

A public operation for other class(es) to use to get myProdFileName attribute value for production history.

- GetProdFileSize

```
EcTInt GetProdFileSize();
```

A public operation for other class(es) to use to get myProdFileSize attribute value production history.

- GetSQFlag

```
RWCString GetSQFlag() const;
```

A public operation for other class(es) to use to get the mySQFlag attribute value.

- GetTime

```
RWCString GetTime() const;
```

A public operation for other class(es) to use to get myTime attribute value.

- GetUR

```
const RWCString GetUR();
```

A public operation for other class(es) to use to get myUR attribute value.

- operator ==

```
EcTBoolean operator ==(const DpPrQaDataGranule& dataGranObj);
```

A public equality operator supports the comparison of two DpPrQaDataGranule objects. If they are equal, return 1; otherwise, return 0.

- operator =

```
DpPrQaDataGranule& operator =(const DpPrQaDataGranule& dataGranObj);
```

A public assignment operator supports the assignment of two or more DpPrQaDataGranule objects.

- PrintMembers

```
EcTVoid PrintMembers();
```

- SetAQFlag

```
EcTVoid SetAQFlag(const RWCString& aqflag);
```

A public operation for other class(es) to use to set the myAQFlag attribute value.

- SetDataFileName

```
EcTVoid SetDataFileName(const RWCString& FileName);
```

A public operation for other class(es) to use to set the myDataFileName attribute value for data granule.

- SetDataFileSize

```
EctVoid SetDataFileSize(const EctInt fileSize);
```

A public operation for other class(es) to use to set myDataFileSize attribute value for data granule.

- SetDate

```
EctVoid setDate(const RWCString& date);
```

A public operation for other class(es) to use to set myDate attribute value.

- SetOQFlag

```
EctVoid SetOQFlag(const RWCString& oqflag);
```

A public operation for other class(es) to use to set the myOQFlag attribute value.

- SetProdFileName

```
EctVoid SetProdFileName(const RWCString& fileName);
```

A public operation for other class(es) to use to set the myProdFileName attribute value for production history.

- SetProdFileSize

```
EctVoid SetProdFileSize(const EctInt fileSize);
```

A public operation for other class(es) to use to set myProdFileSize attribute value for production history.

- SetSQFlag

```
EctVoid SetSQFlag(const RWCString& sqflag);
```

A public operation for other class(es) to use to set the mySQFlag attribute value.

- SetTime

```
EctVoid SetTime(const RWCString& Time);
```

A public operation for other class(es) to use to set myTime attribute value.

- SetUR

```
EctVoid SetUR(const RWCString& urId);
```

A public operation for other class(es) to use to set the myUR attribute value.

7.3.6.3.1 DpPrQaMonitor Class

Overview:

The DpPrQaMonitor is what the Quality Assurance (QA) position uses to query all data granules for a particular data type within a date interval, retrieve data granules from Data Server to local disk, and to view data products created with QA metadata using EOSVIEW tool. The operator can update the QA metadata used to produce the data.

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

`DpPrQaMonitor();`

This is the default constructor

`DpPrQaMonitor(EcUtStatus& status);`

This is the alternate constructor. It gets all necessary environment variables

`~DpPrQaMonitor();`

This is the default constructor.

Operations:

- `InitDataTypesPool`

`EcUtStatus InitDataTypesPool();`

default destructor This operation invokes PLANG operation to initialize the pool of all available Data Types from database.

- `QueryData`

`EcUtStatus QueryData(const RWCString& dType, const RWDate& strtDate, const RWDate& stopDate, <any> ...);`

This operation allows the QA operator to query data granules from DataServer according to data type and date interval.

- `RetrieveData`

```
EcUtStatus RetrieveData(const RWTValslist& urList, <any> ...);
```

This operation allows the QA operator to retrieve data granules from DataServer to local disk.

- RetrieveProdHist

```
EcUtStatus RetrieveProdHist(const RWTValslist& urList, <any> ...);
```

This operation allows the QA operator to retrieve production history from DataServer to local disk.

- UpdateMetaData

```
EcUtStatus UpdateMetaData(const RWCString& oQFlag, const RWCString& scfQFlag, const RWCString& autoQFlag, const RWTValslist& urList);
```

This operation is used by the QA Monitor position to send request to Data Server to update the operational quality flag of MetaData associated with the given product.

- VisualizeData

```
EcUtStatus VisualizeData(const RWCString& filename);
```

This operation will be used to produce a visual interpretation of the given data product for QA purposes. Depending on the type of data requested, various tools will be invoked to display these images.

7.3.7 Processing_QA_Monitor_GUI Class Category

7.3.7.1 Overview

DpPrQaMmainWindow
myQaMonitorPtr : DpPrQaMonitor* mySelectedDataType : RWCString
create() className() helpMenuHelp() PrintOKPBAActivate() PrintCancelPBAActivate() RetrievePBAActivate() ProdHistPBAActivate() UpdatePBAActivate() QueryPBAActivate() PopulateDTList() SingleClickDT() SelectDataGran() UpdateOKPBAActivate() UpdateCancelPBAActivate() UpdateHelpPBAActivate() fileSelectionBoxOkCB() fileMenuPrintActivate() fileMenuExitActivate() helpMenuOnHelpActivate() helpMenuOnContextActivate() helpMenuOnWindowActivate() helpMenuOnKeysActivate() helpMenuIndexActivate() helpMenuTutorialActivate() helpMenuOnVersionActivate() \$PrintOKPBAActivateCallback() \$PrintCancelPBAActivateCallback() \$RetrievePBAActivateCallback() \$ProdHistPBAActivateCallback() \$UpdatePBAActivateCallback() \$QueryPBAActivateCallback() \$PopulateDTListCallback() \$SingleClickDTCallback() \$SelectDataGranCallback() \$UpdateOKPBAActivateCallback() \$UpdateCancelPBAActivateCallback() \$UpdateHelpPBAActivateCallback() helpMenuOnModeActivate() \$fileSelectionBoxOkCBCallback() \$fileMenuPrintActivateCallback() \$fileMenuExitActivateCallback() \$helpMenuOnHelpActivateCallback() \$helpMenuOnContextActivateCallback() \$helpMenuOnWindowActivateCallback() \$helpMenuOnKeysActivateCallback() \$helpMenuIndexActivateCallback() \$helpMenuTutorialActivateCallback() \$helpMenuOnVersionActivateCallback() \$helpMenuOnModeActivateCallback() PrintStatusMsg() SetUp() GetTopLevelShell()

Figure 7.3.7.1-1 Processing_QA_Monitor_GUI

7.3.7.2 Processing_QA_Monitor_GUI Classes

7.3.7.3 DpPrQaMmainWindow Class

Overview:

Export Control: **Public**

Inheritance Relationships:

Attributes:

myQaMonitorPtr: DpPrQaMonitor*

Begin user code block <private> Class data members

mySelectedDataType: RWCString

_appDefaults: UIAppDefault*

_buttonBox1: Widget

_buttonBox2: Widget

_buttonBox: Widget

_cancelButton: Widget

_clientDataStructs: UICallbackStruct*

Callback client data.

_dataGranuleList: Widget

_dataTypeList: Widget

```
_dateIntervalFrame: Widget  
  
_defaultDpPrQaMmainWindowResources: String*  
Default application and class resources.  
  
_defaultOQFPbutton: Widget  
  
_DpPrQaMmainWindow: Widget  
Widgets created by this class  
  
_ecsMenuBar: Widget  
  
_ecUtDateInterval: EcUtDateInterval*  
Classes created by this class  
  
_exit: Widget  
  
_failOQFPButton: Widget  
  
_fileCascade: Widget  
  
_fileMenu: Widget  
  
_fileSelectionBox: Widget  
  
_form: Widget  
  
_frame: Widget  
  
_helpCascade: Widget  
  
_helpMenu: Widget
```

```
_index: Widget  
  
_initAppDefaults: Boolean  
  
_invOQFPButton: Widget  
  
_messageText: Widget  
  
_notInvOQFPButton: Widget  
  
_onContext: Widget  
  
_onHelp: Widget  
  
_onKeys: Widget  
  
_onMode: Widget  
  
_onVersion: Widget  
  
_onWindow: Widget  
  
_operQFlagValuesMenu: Widget  
  
_operQFlag: Widget  
  
_passOQFPButton: Widget  
  
_printBothtoggleButton: Widget
```

```
_printCancelpushButton: Widget  
  
_printDGtoggleButton: Widget  
  
_printDTtoggleButton: Widget  
  
_printForm: Widget  
  
_printOKpushButton: Widget  
  
_print: Widget  
  
_processCascade: Widget  
  
_processMenuItem1: Widget  
  
_processMenuItem2: Widget  
  
_processMenu: Widget  
  
_prodHistButton: Widget  
  
_pulldownMenu: Widget  
  
_qRUFunc1: Widget  
  
_queryButton: Widget  
  
_radioBox: Widget
```

```
_retrieveButton: Widget  
  
_separator: Widget  
  
_statusLabel: Widget  
  
_statusWindow: Widget  
  
_tabStack: Widget  
  
_tutorial: Widget  
  
_updateButton: Widget  
  
_updateCancelButton: Widget  
  
_updateFormDialog: Widget  
  
_updateHelpButton: Widget  
  
_updateOKButton: Widget  
  
_visualizeFunc2: Widget  
  
_xmDialogShell1: Widget  
  
_xmDialogShell: Widget
```

Constructors and Destructor:

```
DpPrQaMmainWindow(const char* );
```

```
virtual ~DpPrQaMmainWindow();
```

Operations:

- className

```
const char* const className();
```

- create

```
virtual void create();
```

- fileMenuExitActivateCallback

```
static void fileMenuExitActivateCallback(Widget );
```

- fileMenuExitActivate

```
virtual void fileMenuExitActivate(Widget );
```

- fileMenuPrintActivateCallback

```
static void fileMenuPrintActivateCallback(Widget );
```

- fileMenuPrintActivate

```
virtual void fileMenuPrintActivate(Widget );
```

- fileSeclectionBoxOkCBCallback

```
static void fileSeclectionBoxOkCBCallback(Widget );
```

- fileSeclectionBoxOkCB

```
virtual void fileSeclectionBoxOkCB(Widget );
```

- GetTopLevelShell

```
Widget& GetTopLevelShell(Widget& w);  
Begin user code block <protected>  
• helpMenuHelp  
void helpMenuHelp(EcUtCIHelp* );  
Begin user code block <public>  
• helpMenuItemActivateCallback  
static void helpMenuItemActivateCallback(Widget );  
  
• helpMenuItemActivate  
virtual void helpMenuItemActivate(Widget );  
  
• helpMenuOnContextActivateCallback  
static void helpMenuOnContextActivateCallback(Widget );  
  
• helpMenuOnContextActivate  
virtual void helpMenuOnContextActivate(Widget );  
  
• helpMenuOnHelpActivateCallback  
static void helpMenuOnHelpActivateCallback(Widget );  
  
• helpMenuOnHelpActivate  
virtual void helpMenuOnHelpActivate(Widget );  
  
• helpMenuOnKeysActivateCallback  
static void helpMenuOnKeysActivateCallback(Widget );  
  
• helpMenuOnKeysActivate  
virtual void helpMenuOnKeysActivate(Widget );  
  
• helpMenuOnModeActivateCallback
```

- helpMenuOnModeActivate
 - `static void helpMenuOnModeActivate(Widget);`
 - `virtual void helpMenuOnModeActivate(Widget);`
- helpMenuOnVersionActivateCallback
 - `static void helpMenuOnVersionActivateCallback(Widget);`
- helpMenuOnVersionActivate
 - `virtual void helpMenuOnVersionActivate(Widget);`
- helpMenuOnWindowActivateCallback
 - `static void helpMenuOnWindowActivateCallback(Widget);`
- helpMenuOnWindowActivate
 - `virtual void helpMenuOnWindowActivate(Widget);`
- helpMenuTutorialActivateCallback
 - `static void helpMenuTutorialActivateCallback(Widget);`
- helpMenuTutorialActivate
 - `virtual void helpMenuTutorialActivate(Widget);`
- PopulateDTListCallback
 - `static void PopulateDTListCallback(Widget);`
- PopulateDTList
 - `virtual void PopulateDTList(Widget);`
- PrintCancelPBActivateCallback

```
static void PrintCancelPBActivateCallback(Widget );
```

- PrintCancelPBActivate

```
virtual void PrintCancelPBActivate(Widget );
```

- PrintOKPBActivateCallback

```
static void PrintOKPBActivateCallback(Widget );
```

Callbacks to interface with Motif.

- PrintOKPBActivate

```
virtual void PrintOKPBActivate(Widget );
```

These virtual functions are called from the private callbacks or event handlers intended to be overridden in derived classes to define actions

- PrintStatusMsg

```
void PrintStatusMsg(EcTChar* , ... );
```

Class private operations void PrintStatusMsg(va_list);

- ProdHistPBActivateCallback

```
static void ProdHistPBActivateCallback(Widget );
```

- ProdHistPBActivate

```
virtual void ProdHistPBActivate(Widget );
```

- QueryPBActivateCallback

```
static void QueryPBActivateCallback(Widget );
```

- QueryPBActivate

```
virtual void QueryPBActivate(Widget );
```

- RetrievePBActivateCallback

```
static void RetrievePBActivateCallback(Widget );
```

- RetrievePBActivate

```
virtual void RetrievePBActivate(Widget );
```

- SelectDataGranCallback

```
static void SelectDataGranCallback(Widget );
```

- SelectDataGran

```
virtual void SelectDataGran(Widget );
```

- SetUp

```
EcUtStatus SetUp(const Widget& parent);
```

- SingleClickDTCallback

```
static void SingleClickDTCallback(Widget );
```

- SingleClickDT

```
virtual void SingleClickDT(Widget );
```

- UpdateCancelPBActivateCallback

```
static void UpdateCancelPBActivateCallback(Widget );
```

- UpdateCancelPBActivate

```
virtual void UpdateCancelPBActivate(Widget );
```

- UpdateHelpPBActivateCallback

```
static void UpdateHelpPBActivateCallback(Widget );
```

- UpdateHelpPBActivate

```
virtual void UpdateHelpPBActivate(Widget );
```

- UpdateOKPBAActivateCallback

```
static void UpdateOKPBAActivateCallback(Widget );
```

- UpdateOKPBAActivate

```
virtual void UpdateOKPBAActivate(Widget );
```

- UpdatePBActivateCallback

```
static void UpdatePBActivateCallback(Widget );
```

- UpdatePBActivate

```
virtual void UpdatePBActivate(Widget );
```

7.3.8 Processing_Resource_Management Class Category

7.3.8.1 Overview

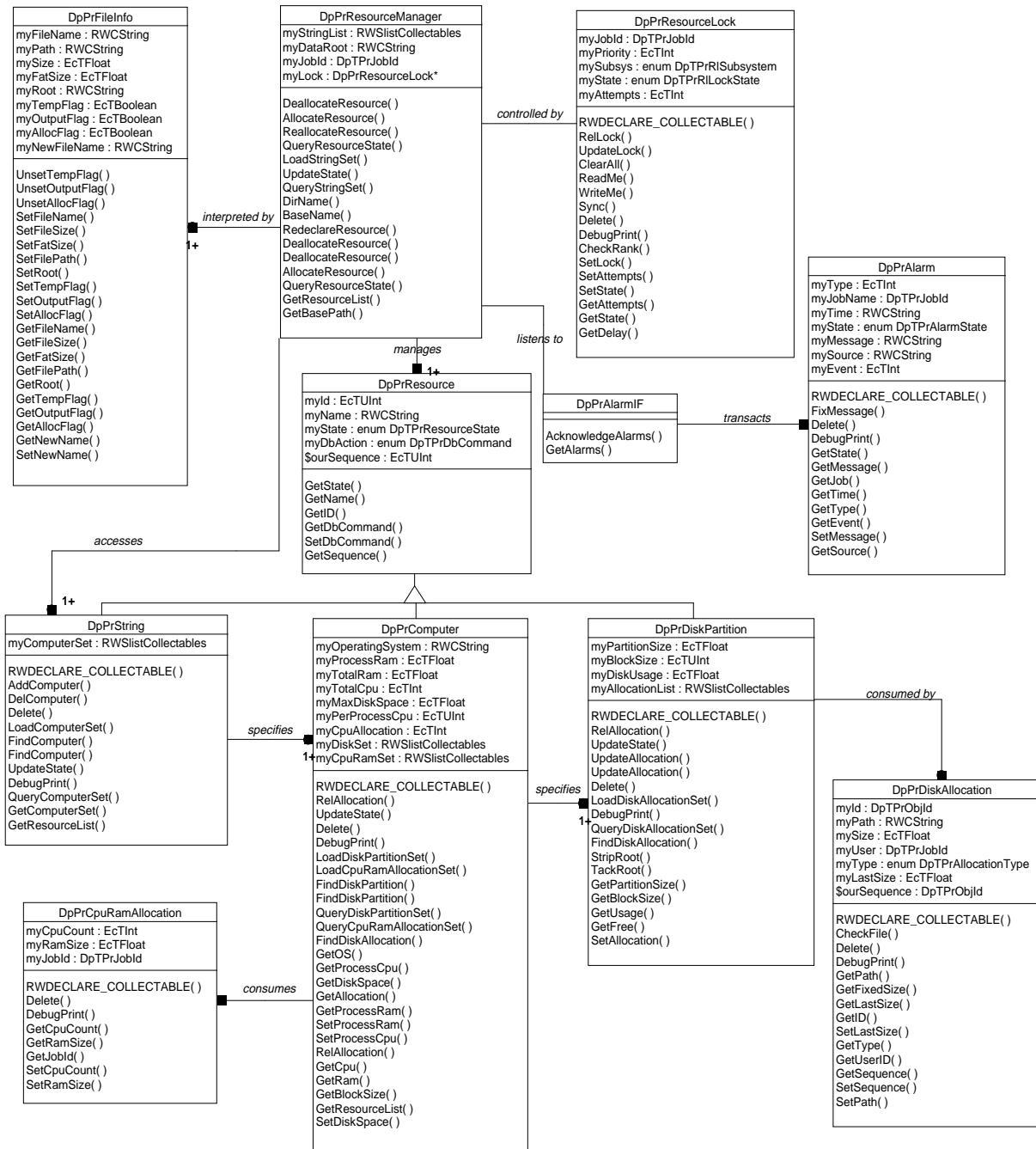


Figure 7.3.8.1-1 Resource_Management

7.3.8.2 Processing_Resource_Management Classes

7.3.8.3 DpPrAlarmIF Class

Overview:

This class is used to interface with the AutoSys Alarm Manager table in order to monitor the alarm traffic generated by Data Processing applications.

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

`DpPrAlarmIF();`

This constructor will create an instance of the AlarmIF object

`DpPrAlarmIF(EcUtStatus& alarmStatus);`

This constructor will create an instance of the AlarmIF object and establish the AUTOSYS DB connection.

`~DpPrAlarmIF();`

Destroy the DpPrAlarmIF object instance.

Operations:

- AcknowledgeAlarms

`EcUtStatus AcknowledgeAlarms(const EcTChar& indicator, const RWTValslist& ackAlarmList);`

Issue a modified alarm message to signify completion of alarm processing.

- GetAlarms

`EcUtStatus GetAlarms(const EcTChar& indicator, <any> ...);`

Get list of recently generated alarms.

7.3.8.3.1 DpPrAlarm Class

Overview:

This class is used to retrieve and mark AutoSys alarms that are deposited into the Autosys database by other DPS applications.

Export Control: Public

Inheritance Relationships:

Attributes:

myEvent: EcTInt

The Trace Event Number

myJobName: DpTPrJobId

AutoSys job name; a.k.a. DPR identifier

myMessage: RWCString

Alarm Comment

mySource: RWCString

Originating host

myState: enum DpTPrAlarmState

State of alarm

myTime: RWCString

Alarm timestamp

myType: EcTInt

Type of Alarm

Constructors and Destructor:

DpPrAlarm();

This constructor will create an instance of the Alarm object

~DpPrAlarm();

Destroy the DpPrAlarm object instance.

Operations:

- DebugPrint

```
EcTVoid DebugPrint();
```

number of row(s) deleted This operation prints to standard out the values of the object attributes .

- Delete

```
EcUtStatus Delete(EcTInt& rowsDeletedAlarm);
```

DpPrAlarm class object This operation calls DeleteRows in DbInterface object to delete row(s) in AutoSys alarm table. It emulates the SQL statement: 'Delete from autosys.alarm where "job_name" = this->myJobName'

- FixMessage

```
EcUtStatus FixMessage(const EcTChar& symbol);
```

This operation overwrites the existing AutoSys alarm message with an altered message that indicates the alarm has been processed.

- GetEvent

```
EcTInt GetEvent();
```

This operation returns the AutoSys Event number

- GetJob

```
DpTPrJobId GetJob();
```

This operation returns the job name that issued the alarm

- GetMessage

```
RWCString GetMessage();
```

This operation returns the alarm message

- GetSource

```
RWCString GetSource();
```

unprocessed message indicator This operation returns the host machine where the alarm was generated

- GetState

```
enum DpTPrAlarmState GetState();
```

This operation returns the alarm state

- GetTime

```
RWCString GetTime();
```

This operation returns the timestamp on the alarm

- GetType

```
EcTInt GetType();
```

This operation returns the alarm type

- RWDECLARE_COLLECTABLE

```
int RWDECLARE_COLLECTABLE(DpPrAlarm );
```

This class is used to retrieve and mark AutoSys alarms that are deposited into the Autosys database by other DPS applications.

- SetMessage

```
EcUtStatus SetMessage(const RWCString& newComment);
```

This operation sets a new alarm message

7.3.8.3.2 DpPrComputer Class

Overview:

This class is used to represent the set of computer hardware that is being used for science software processing within the Processing System. All management activities for controlling the use of processing resources are performed by this class.

Export Control: Public

Inheritance Relationships:

Inherits from [DpPrResource](#)

Attributes:

myCpuAllocation: EcTInt

This attribute defines the number of processors which are currently allocated to the processing of PGEs on this platform. This value is periodically adjusted to account for the allocation and deallocation of processing resources for PGEs. This value cannot exceed the total number of CPUs that defined for this platform.

myCpuRamSet: RWSlistCollectables

This attribute points to the set of objects which represent the CPU and RAM allocations on a per-job basis

myDiskSet: RWSlistCollectables

This attribute points to the set of objects which represent the attached storage devices.

myMaxDiskSpace: EcTFloat

This attribute defines the total amount of local disk space which is available for PGE processing. The amount of disk space that is used by the system is not included in this value.

myOperatingSystem: RWCString

Disk connection category

Private attributes

This attribute defines the machine type and the current version of the operating system which controls it.

myPerProcessCpu: EcTUInt

This attribute defines the per process limit on processing time which may be granted to an individual process. This limit is imposed by the underlying system and may only be increased up to some system defined limit. Any PGE process which exceeds this limit will not run to completion on this platform.

myProcessRam: EcTFloat

This attribute tracks the sum of all RAM allocations by all current processes. Note that this value may exceed the actual total RAM available (myTotalRam) because jobs may share the available memory.

myTotalCpu: EcTInt

This attribute represents the actual number of individual processors which may be applied to the processing of one or more PGEs. Only individual processors may be allocated to the processing effort for a single PGE.

myTotalRam: EcTFloat

This attribute defines the total RAM configuration for the object instance. This value is used as a coarse gauge when selecting a computing platform, if a specific platform is not chosen, whereas the per process RAM setting is considered to be a hard limit.

Constructors and Destructor:

```
DpPrComputer(EcUtStatus& rsrcStatus, DpTPrMachineId name,  
RWCString os, EcTUInt memory, EcTInt processing, EcTUInt  
stringId, enum DpTPrDiskConnect storage, enum DpTPrResourceState  
state);
```

Data Processing Request job ID This operation creates a DpPrComputer instance

DpPrComputer();

This default constructor will be used to create an empty Computer object, before the DataBase Interface tools are called to transfer Computer information from the persistent storage area.

~DpPrComputer();

Destroy the DpPrComputer object instance and the associated collection of DpPrDiskPartition resource objects.

Operations:

- DebugPrint

```
EcTVoid DebugPrint();
```

Number of row(s) deleted This function prints to standard out the values of the object attributes

- Delete

```
EcUtStatus Delete(EcTInt& rowsDeletedResource);
```

Computer resource object This operation calls DeleteRows in DbInterface object to delete row(s) in RSC_COMPUTER table. It emulates the Delete SQL statement: 'Delete from RSC_COMPUTER where "COMPUTER_ID" = this->my.Id'

- FindDiskAllocation

```
EcTBoolean FindDiskAllocation(RWCString fileName, RWCString& filePath);
```

This routine will search the DpPrDiskPartition set for an object containing a matching file name.

- FindDiskPartition

```
EcUtStatus FindDiskPartition(RWCString partitionRoot, EcTFloat freeSpace, RWCString& partitionPath, DpPrDiskPartition** partitionObjPtr);
```

DpPrDiskPartition pointer This routine will search the list of DpPrDiskPartition objects associated with this DpPrComputer object for an element whose current free space equals or exceeds the value freeSpace. If a match is found, a pointer to the specified DpPrDiskPartition object is returned. Otherwise the pointer is set to zero.

- FindDiskPartition

```
EcUtStatus FindDiskPartition(RWCString partitionRoot, RWCString pathName, RWCString& partitionPath, DpPrDiskPartition** partitionObjPtr);
```

This routine will search the list of DpPrDiskPartition objects associated with this DpPrComputer object for an element whose absolute path name is in the parent path of pathName. If a match is found, a pointer to the specified DpPrDiskPartition object is returned. Otherwise the pointer is set to zero.

- GetAllocation

```
EcTInt GetAllocation();
```

View category: FREE/USER/TOTAL space The current processor allocation level is returned to the calling process.

- GetBlockSize

EcTUInt GetBlockSize();

This operation gets the value of the Block Size attribute from the first associated DpPrDiskPartition object and returns it to calling process.

- GetCpuLimit

EcTUInt GetCpuLimit();

The per process CPU time limit is acquired from the designated platform. This value may be increased by authorized processes.

- GetCpu

EcTUInt GetCpu();

The value of the Total Cpu attribute is returned to the calling process.

- GetDevices

RWSlistCollectables GetDevices(enum DpTPrDiskConnect range);

Returns a reference to a list of associated Disk Device objects. These objects are constructed and initialized during the process. Either local, or both local and mounted devices are returned based on the designated connection type.

- GetDiskSpace

EcTFloat GetDiskSpace(enum DpTPrQueryDisk view);

Depending on the designator used to define the perspective, the requested disk space value is returned to the calling process. The Max Disk Space attribute normally contains the total disk space which can be allocated for user needs and therefore may not be used as the return value unless the designator so indicates.

- GetOS

RWCString GetOS();

Amount of memory required, in bytes The value of the Operating System attribute is returned to the calling process.

- GetProcessCpu

EcTUInt GetProcessCpu();

The value of the Per Process Cpu attribute is returned to the calling process.

- GetProcessRam

```
EcTFloat GetProcessRam();
```

The current value of the Process Ram attribute is returned to the calling process.

- GetRamLimit

```
EcTUInt GetRamLimit();
```

The per process Heap limit is obtained from the designated platform. This value will be the soft limit unless some authorized process increased the value.

- GetRam

```
EcTUInt GetRam();
```

The value of the Total Ram attribute is returned to the calling process.

- GetResourceList

```
EcUtStatus GetResourceList(RWSlistCollectables& resourceSet,  
enum DpTPrResourceType type);
```

This operation generates a list of all objects of the specified resource type currently found in the resource tables

- LoadCpuRamAllocationSet

```
EcUtStatus LoadCpuRamAllocationSet(void );
```

This routine will create all instances for the DpPrCpuRamAllocation class associated with this DpPrComputer object, and regenerate the resource objects from the persistent database storage.

- LoadDiskPartitionSet

```
EcUtStatus LoadDiskPartitionSet(void );
```

Available disk space This routine will create all object instances for the DpPrDiskPartition class associated with this DpPrComputer object, and regenerate the resource objects from the persistent database storage.

- QueryCpuRamAllocationSet

```
EcTVoid QueryCpuRamAllocationSet();
```

This routine will step through the list of DpPrCpuRamAllocation objects associated with this DpPrComputer object. Information is output for each element in the list.

- QueryDiskPartitionSet

```
EcTVoid QueryDiskPartitionSet();
```

DpPrDiskPartition pointer This routine will step through the list of DpPrDiskPartition objects associated with this DpPrComputer object. Information is output for each element in the list.

- RelAllocation

```
EcUtStatus RelAllocation(DpTPrJobId job, EcTInt cpuCount,  
EcTFloat ramSize);
```

New limit, in cycles This operation will release the allocation of CPUs and RAM for the job specified.

- RelAllocation

```
EcUtStatus RelAllocation(DpTPrJobId job);
```

Amount of RAM to release, in bytes This operation will release the total allocation of CPUs and RAM for the job specified.

- RWDECLARE_COLLECTABLE

```
int RWDECLARE_COLLECTABLE(DpPrComputer );
```

This class is used to represent the set of computer hardware that is being used for science software processing within the Processing System. All management activities for controlling the use of processing resources are performed by this class.

- SetAllocation

```
EcUtStatus SetAllocation(DpTPrJobId job, EcTInt cpuCount,  
EcTFloat ramSize);
```

Define the amount of processing power needed to run the job. Unless the requested amount resources are unavailable, an allocation will be made on behalf of the designated job. job - The Processing Subsystem assigned identifier for the job cpuCount - The number of processors required for the job ramSize - The amount of memory required for the job, in bytes

- SetDiskSpace

```
EcUtStatus SetDiskSpace(EcTFloat newSize);
```

Resource type The Max Disk Space attribute, normally established by accumulating the individual Disk Partition totals (via GetDiskSpace), may be overridden by the use of the operation.

- SetProcessCpu

```
EcUtStatus SetProcessCpu(EcTUInt newLimit);
```

New limit, in bytes The soft limit imposed by the system may be increased up to the predefined hard limit; authorized users may increase this value beyond the hard limit.

- SetProcessRam

```
EcUtStatus SetProcessRam(EcTFloat newLimit);
```

The default soft limit, that is imposed by the system, can be increased up to the hard limit for that system; authorized users may increase the value beyond the hard limit. newLimit - The new limit expressed in terms of bytes

- **UpdateState**

`EcUtStatus UpdateState();`

Current resource state Acquire the current variable configuration settings for the object and populate those attribute fields. This data consists of the per process CPU and memory settings which may be modified by authorized processes. Also, the current user available disk space is updated. This operation also triggers the state update operation for each resource sub-object to get a current assessment of the entire resource pool.

7.3.8.3.3 DpPrCpuRamAllocation Class

Overview:

This class is used to maintain the individual records of CPU and RAM usage, which are used to maintain the integrity of processor and memory allocations and deallocations. These records are uniquely associated with a particular computer. Memory allocations are actually pseudo-allocations, in that the total of all memory allocated may exceed the amount of physical memory on the associated computer.

Export Control: Public

Inheritance Relationships:

Attributes:

`myCpuCount: EcTInt`

Private attributes

This attribute holds the number of processors allocated to the specified job from the specified DpPrComputer object.

`myJobId: DpTPrJobId`

This attribute holds the value of the identifier specified in the original allocation request and represents the job that is associated with this CPU and RAM allocation

`myRamSize: EcTFloat`

This attribute holds the RAM size in Kb allocated to the specified job from the specified DpPrComputer object.

Constructors and Destructor:

`DpPrCpuRamAllocation();`

This default constructor will be used to create an empty CPU/RAM Allocation object, before the DataBase Interface tools are called to transfer CPU and RAM allocation information from the persistent storage area.

```
DpPrCpuRamAllocation(EcUtStatus& rsrcStatus, EcTInt cpuCount,  
EcTFloat ramSize, DpTPrJobId jobId, EcTUInt computerId);
```

This constructor will be used to create the object, as well as define the values of the attributes.
jobId - The user for whom the allocation is being made initially
computerId - The ECS defined unique resource identifier which represents the computer which is associated with this CPU/
RAM allocation
id - The ECS defined unique resource identifier which represents this CPU/
RAM allocation
rsrcStatus - The status condition return object: OK FAILED

```
~DpPrCpuRamAllocation();
```

This deallocator will be used to remove the object.

Operations:

- DebugPrint

```
EcTVoid DebugPrint();
```

This function prints to standard out the values of the object attributes

- Delete

```
EcUtStatus Delete(EcTInt& rowsDeleted);
```

CpuRamAllocation class object This operation calls DeleteRows in DbInterface object to delete row(s) in RSC_CPU_RAM_ALLOCATION table. It emulates the Delete SQL statement:
'Delete from RSC_CPU_RAM_ALLOCATION where
"CPU_RAM_ALLOCATION_JOB_ID" = this->myJobId' rowsDeleted - number of row(s) deleted

- GetCpuCount

```
EcTInt GetCpuCount();
```

ID of associated DpPrComputer object Retrieve the value of the myCpuCount attribute, which maintains the value of the original CPU resource allocation.

- GetJobId

```
DpTPrJobId GetJobId();
```

Retrieve the job identifier responsible for originating this allocation instance.

- GetRamSize

```
EcTFloat GetRamSize();
```

Retrieve the value of the myRamSize attribute, which maintains the value of the RAM resource pseudo-allocation.

- RWDECLARE_COLLECTABLE

```
int RWDECLARE_COLLECTABLE(DpPrCpuRamAllocation );
```

This class is used to maintain the individual records of CPU and RAM usage, which are used to maintain the integrity of processor and memory allocations and deallocations. These records are uniquely associated with a particular computer. Memory allocations are actually pseudo-allocations, in that the total of all memory allocated may exceed the amount of physical memory on the associated computer.

- SetCpuCount

```
void SetCpuCount(EcTInt cpuCount);
```

Set the value of the myCpuCount attribute, which tracks the number of processors currently allocated to the specified job.

- SetRamSize

```
void SetRamSize(EcTFloat ramSize);
```

Number of processors allocated Set the value of the myRamSize attribute, which tracks the amount of memory currently allocated to the specified job.

7.3.8.3.4 DpPrDiskAllocation Class

Overview:

This class is used to maintain the individual records of disk storage usage which are used to maintain the integrity of storage allocations and deallocations. These records are uniquely associated with a particular disk partition (i.e., file system) and computer.

Export Control: Public

Inheritance Relationships:

Attributes:

myId: DpTPrObjId

Private attributes

This attribute uniquely defines an object of this class. Its use is primarily for programming convenience. (A computer, partition and path uniquely identify a granule in human terms)

myLastSize: EcTFloat

This attribute represents the latest size recorded for the file specified by the Path attribute. It will be used to compare against the original allocation size to determine if a file is exceeding its expected size.

myPath: RWCString

This attribute defines the entire directory path to the file for which this allocation is being made.

mySize: EcTFloat

This attribute represents the size specified for the original allocation request. It will be used to compare against the actual size of the file represented by this allocation to check for an unexpected increase in size.

myType: enum DpTPrAllocationType

This attribute defines whether the disk space was allocated for system files or user i.e., science software files. It is an internal identifier used to uniquely track individual allocations. The User attribute, conversely, is not unique in that there may be many allocations for a single job.

myUser: DpTPrJobId

This attribute holds the value of the identifier specified in the original allocation request and represents the job that is associated with the file defined by the Path attribute.

ourSequence: DpTPrObjId

This attribute is a unique resource ID generator for all of the derived resource classes.

Constructors and Destructor:

```
DpPrDiskAllocation(EcUtStatus& rsrcStatus, RWCString path,  
DpTPrJobId job, EcTFloat size, DpTPrObjId computerId, DpTPrObjId  
deviceId, enum DpTPrAllocationType type);
```

This constructor will be used to create the object, as well as define the values of the static attributes job, type, user, size and path; only the lastSize attribute is considered to be dynamic.

```
DpPrDiskAllocation();
```

This default constructor will be used to create an empty Disk Allocation object, before the DataBase Interface tools are called to transfer Disk Allocation information from the persistent storage area.

```
~DpPrDiskAllocation();
```

This deallocator will be used to remove the object.

Operations:

- CheckFile

```
EcTBoolean CheckFile();
```

This operation provides the means to check on the existence of the file, associated with this allocation, before actually releasing the allocation.

- DebugPrint

```
EcTVoid DebugPrint();
```

Number of row(s) deleted This function prints to standard out the values of the object attributes

- Delete

```
EcUtStatus Delete(EcTInt& rowsDeletedAllocation);
```

DiskAllocation class object This operation calls DeleteRows in DbInterface object to delete row(s) in RSC_DISK_ALLOCATION table. It emulates the Delete SQL statement: 'Delete from RSC_DISK_ALLOCATION where "DISK_ALLOCATION_ID" = this->myId'

- GetFixedSize

```
EcTFloat GetFixedSize();
```

Retrieve the value of the size attribute, which maintains the value of the original resource allocation.

- GetID

```
DpTPrObjId GetID();
```

Allocation category Retrieve the unique, sequence generated, identifier which is defined by the attribute ID.

- GetLastSize

```
EcTFloat GetLastSize();
```

This operation retrieves the value of the lastSize attribute, which is updated on a periodic basis to reflect the actual amount of disk space that is being consumed by the allocated file.

- GetPath

```
RWCString GetPath();
```

Retrieve the full filepath to the entity defined by this instance of the object.

- GetSequence

```
DpTPrObjId GetSequence();
```

Retrieve the next available id for a new disk allocation object.

- GetType

```
enum DpTPrAllocationType GetType();
```

This operation identifies this object as being a normal user allocation, or a non-user allocation which implies that the disk space held by the allocation cannot be used directly for science processing purposes.

- GetUserID

```
DpTPrJobId GetUserID();
```

Retrieve the job identifier responsible for originating this allocation instance.

- RWDECLARE_COLLECTABLE

```
int RWDECLARE_COLLECTABLE(DpPrDiskAllocation );
```

This class is used to maintain the individual records of disk storage usage which are used to maintain the integrity of storage allocations and deallocations. These records are uniquely associated with a particular disk partition (i.e., file system) and computer.

- SetLastSize

```
EcTVoid SetLastSize(EcTFloat newSize);
```

This operation updates the value of the lastSize attribute.

- SetPath

```
EcTVoid SetPath(const RWCString newPath);
```

<< NCR ECSSed04997 Establish a new device path, to the entity defined for this instance of the object, which reflects the name of the physical object for which disk space was previously allocated. The form of a device path follows: 'Partition-name/Directory-names/File-name'

- SetSequence

```
EcUtStatus SetSequence();
```

Initialize the sequence generator from the current allocation level found in the database.

7.3.8.3.5 DpPrDiskPartition Class

Overview:

This class is used to represent the set of disk storage devices that are being used to contain the files needed to run a collective set of software known as a PGE. This includes input and output data files used and produced by the science software, as well as the executable files which comprise the PGE. All management activities for controlling the use of the disk resources are performed by this class.

Export Control: Public

Inheritance Relationships:

Inherits from [DpPrResource](#)

Attributes:

myAllocationList: RWSlistCollectables

This attribute identifies the reference to a list of disk resource

myBlockSize: EcTUIInt

This attribute holds the block size, which is a fixed value imposed by the particular operating system. It should be used to convert file sizes to units of bytes.

myDiskUsage: EcTFloat

This derived attribute maintains the current amount of disk resources which are presently allocated for the use of science processing.

myPartitionSize: EcTFloat

Private attributes

This attribute holds the amount of disk space allocated to the file system. This value is fixed during file system creation

Constructors and Destructor:

```
DpPrDiskPartition(EcUtStatus& rsrcStatus, RWCString root,  
EcTUInt computerId, EcTFloat partitionSize, enum  
DpTPrResourceState state);
```

This constructor will be used to create the object along with its identification and initial state. Following object creation, all associated system allocations will be created and linked to this object. The actual partition size and current allocation amounts will be initialized.

```
DpPrDiskPartition();
```

NCR ECSed04997 >> This default constructor will be used to create an empty Disk Partition. This default constructor will be used to create an empty Disk Partition object, before the DataBase Interface tools are called to transfer Disk Partition information from the persistent storage area.

```
~DpPrDiskPartition();
```

Destroy the DpPrDiskPartition object instance and the associated collection of DpPrDiskAllocation resource objects.

Operations:

- DebugPrint

```
EcTVoid DebugPrint();
```

ID of associated DpPrComputer object This function prints to standard out the values of the object attributes

- Delete

```
EcUtStatus Delete(EcTInt& rowsDeletedResource);
```

Disk Partition object This operation calls DeleteRows in DbInterface object to delete row(s) in RSC_DISK_PARTITION table. It emulates the Delete SQL statement: 'Delete from RSC_DISK_PARTITION where "DISK_PARTITION_ID" = this->myId' rowsDeletedResource - number of row(s) deleted

- FindDiskAllocation

```
EcTBoolean FindDiskAllocation(RWCString fileName, RWCString& filePath);
```

This routine will search the DpPrDiskAllocation set for an object containing a matching file name.

- GetBlockSize

```
EcTUInt GetBlockSize();
```

The value of the Block Size attribute is returned to the calling process.

- GetFree

```
EcTFloat GetFree();
```

Compute the total amount of unused space for this partition which is available for immediate allocation.

- GetPartitionSize

```
EcTFloat GetPartitionSize();
```

Current resource state The value of the Partition Size attribute is returned to the calling process.

- GetUsage

```
EcTFloat GetUsage();
```

The current disk allocation amount is returned.

- LoadDiskAllocationSet

```
EcUtStatus LoadDiskAllocationSet(EcTUInt computerId);
```

Number of row(s) deleted This routine will create all object instances for the DpPrDiskAllocation class associated with this DpPrDiskPartition object, and regenerate the resource objects from the persistent database storage. computerId - The ID of the associated DpPrComputer object

- QueryDiskAllocationSet

```
EcTVoid QueryDiskAllocationSet();
```

This routine will step through the list of DpPrDiskAllocation objects associated with this DpPrDiskPartition object. Information is output for each element in the list.

- RelAllocation

```
EcUtStatus RelAllocation(EcTFloat& size, DpTPrJobId job, RWCString filePath);
```

Allocation type: System/ECS/Temp Individual file allocations are released and the final size of the allocation returned. job - the job ID filePath - the full pathname of the file to deallocate

- RWDECLARE_COLLECTABLE

```
int RWDECLARE_COLLECTABLE(DpPrDiskPartition );
```

This class is used to represent the set of disk storage devices that are being used to contain the files needed to run a collective set of software known as a PGE. This includes input and output data files used and produced by the science software, as well as the executable files which comprise the PGE. All management activities for controlling the use of the disk resources are performed by this class.

- SetAllocation

```
EcUtStatus SetAllocation(EcTFloat size, DpTPrJobId job,  
RWCString filePath, EcTUInt computerId, enum DpTPrAllocationType  
type);
```

Individual resource allocations are generated for science processing uses. The input allocation amount is the reserved amount. The actual amount used by this allocation may be slightly more than the reserved amount to allow for efficient storage.

- SetSysAllocation

```
EcUtStatus SetSysAllocation();
```

During initialization of the object, an initial set of system allocations will be created to account for system and software usage of this partition. This value is assumed to be static for the current configuration of the resources.

- StripRoot

```
EcTBoolean StripRoot(RWCString rootName, RWCString& pathName);
```

Path to the file, if found, unset otherwise This operation attempts to remove the supplied leading root path prefix from the supplied path. It checks first to make sure this is possible.

- TackRoot

```
EcTVoid TackRoot(RWCString rootName, RWCString& pathName);
```

Path name (input/output) This operation prepends the supplied leading root path prefix to the the supplied path. No checking is done.

- UpdateAllocation

```
EcUtStatus UpdateAllocation(RWCString devicePath, RWCString  
fileName, EcTFloat newSize);
```

The disposition of a particular allocation is determined by comparing the new use of resources with the amount originally allocated. If the new allocation amount exceeds the current value, then the value will be updated.

- UpdateAllocation

```
EcUtStatus UpdateAllocation(RWCString devicePath, RWCString  
fileName, RWCString newName);
```

<< NCR ECSSed04997 To correct the allocation record once the actual physical file name is known, this operation can be used to replace the temporary identifier currently used to identify the allocation record.

- UpdateState

EcUtStatus UpdateState();

Path name of file to deallocate The current allocation amounts are derived from the associated allocations in order to update the allocation attributes. This operation will also trigger the state update operation for each resource sub-object to get a current assessment of the entire resource pool.

7.3.8.3.6 DpPrFileInfo Class

Overview:

Export Control: **Public**

Inheritance Relationships:

Attributes:

myAllocFlag: EcTBoolean

This attribute contains the file allocated indicator flag. By default, it is set to EcDFalse. Allocation calls should set it to EcDTrue. Deallocation calls should set it to EcDFalse.

myFatSize: EcTFloat

This attribute contains the size limit of the allocation in Mbytes.

myFileName: RWCString

This attribute contains the name of the file allocation. It may optionally include a relative or absolute path.

myNewFileName: RWCString

<< NCR ECSSed04997 This attribute contains the new identifier to use for the allocation object. The previous identifier MUST be replaced with the new one in order to maintain knowledge of the actual physical file.

myOutputFlag: EcTBoolean

This attribute contains the output file indicator flag. By default, it is set to EcDFalse

myPath: RWCString

This attribute contains the full path of file allocation. It is always an absolute path.

myRoot: RWCString

This attribute contains the partition root directory. It is always an absolute path.

mySize: EcTFloat

This attribute contains the size of the original allocation in Mbytes.

myTempFlag: EcTBoolean

This attribute contains the temporary file indicator flag. By default, it is set to EcDFalse

Constructors and Destructor:

DpPrFileInfo();

This class tracks attributes that are used in calls to allocate and deallocate files. It is used in linked list form, in the calling sequences for members of the class DpPrResourceManager.
Default constructor for the DpPrFileInfo class

DpPrFileInfo(const DpPrFileInfo& pathObject);

Construct an instance of DpPrFileInfo based on an existing instance.

~DpPrFileInfo();

Destructor for the DpPrFileInfo class

Operations:

- GetAllocFlag

EcTBoolean GetAllocFlag();

Retrieve the AllocFlag attribute for an object instance.

- GetFatSize

EcTFloat GetFatSize();

Retrieve the FatSize attribute for an object instance.

- GetFileName

RWCString GetFileName();

Retrieve the FileName attribute for an object instance.

- GetFilePath

RWCString GetFilePath();

Retrieve the Path attribute for an object instance.

- GetFileSize

```
EcTFloat GetFileSize();
```

Retrieve the Size attribute for an object instance.

- **GetNewName**

```
RWCString GetNewName();
```

<< NCR ECSed04997 Retrieve the New file name attribute for the current instance of the object.

- **GetOutputFlag**

```
EcTBoolean GetOutputFlag();
```

Retrieve the OutputFlag attribute for an object instance.

- **GetRoot**

```
RWCString GetRoot();
```

Retrieve the Root attribute for an object instance.

- **GetTempFlag**

```
EcTBoolean GetTempFlag();
```

Retrieve the TempFlag attribute for an object instance.

- **operator ==**

```
EcTBoolean operator ==(const DpPrFileInfo& pathObject);
```

Rvalue in an assignment Test for equality with another object instance

- **operator =**

```
const DpPrFileInfo& operator =(const DpPrFileInfo& pathObject);
```

Existing DpPrFileInfo object Assignment operator for class DpPrFileInfo

- **SetAllocFlag**

```
EcTVoid SetAllocFlag();
```

Sets the file allocated flag.

- **SetFatSize**

```
EcTVoid SetFatSize(EcTFloat fatSize);
```

File size in bytes Set the FatSize attribute for an object instance.

- **SetFileName**

```
EcTVoid SetFileName(const RWCString& fileName);
```

Object instance for compare Set the FileName attribute for an object instance.

- **SetFilePath**

```
EcTVoid SetFilePath(const RWCString& filePath);
```

File size limit in bytes Set the Path attribute for an object instance.

- SetFileSize

```
EcTVoid SetFileSize(EcTFloat fileSize);
```

File Name Set the Size attribute for an object instance.

- SetNewName

```
EcTVoid SetNewName(const RWCString& newFileName);
```

Set the New file name attribute for an object instance.

- SetOutputFlag

```
EcTVoid SetOutputFlag();
```

Sets the output file flag.

- SetRoot

```
EcTVoid SetRoot(const RWCString& root);
```

File path Set the Root attribute for an object instance.

- SetTempFlag

```
EcTVoid SetTempFlag();
```

Root partition path Sets the temporary file flag.

- UnsetAllocFlag

```
EcTVoid UnsetAllocFlag();
```

Unsets the file allocated flag.

- UnsetOutputFlag

```
EcTVoid UnsetOutputFlag();
```

Unsets the output file flag.

- UnsetTempFlag

```
EcTVoid UnsetTempFlag();
```

Unsets the temporary file flag.

7.3.8.3.7 DpPrResourceLock Class

Overview:

This class is used to implement locking of the Resource Management database tables on a per-job basis. This is necessary in order to maintain the integrity of the information contained there.

Export Control: Public

Inheritance Relationships:

Attributes:

`myAttempts: EcTInt`

This attribute tracks the number of attempts by the job to get a lock on the Resource Management tables.

`myJobId: DpTPrJobId`

Private attributes

This attribute represents the job ID of the associated Data Processing Request

`myPriority: EcTInt`

This attribute represents the job priority. It used to help determine which job will be granted the next lock.

`myState: enum DpTPrRlLockState`

This attribute is an enumerated type representing the lock state. It may take on three possible values: waiting, locked, released

`mySubsys: enum DpTPrRlSubsystem`

This attribute is an enumerated type representing the subsystem using the lock. The default subsystem is Resource Management.

Constructors and Destructor:

`DpPrResourceLock();`

This constructor will create an instance of the ResourceLock object

`DpPrResourceLock(EcUtStatus& returnStatus, DpTPrJobId job, enum DpTPrRlSubsystem subsys);`

This constructor will create an instance of the ResourceLock object

`~DpPrResourceLock();`

Destroy the DpPrResourceLock object instance.

Operations:

- `CheckRank`

`EcUtStatus CheckRank(EcTBoolean& ranksFirst);`

This operation performs a test on the lock table to determine if the current job (DPR) is eligible to receive the lock.

- ClearAll

```
EcTVoid ClearAll();
```

Current access state This operation clears all private attributes.

- DebugPrint

```
EcTVoid DebugPrint();
```

Number of row(s) deleted This operation prints to standard out the values of the object attributes

- Delete

```
EcUtStatus Delete(EcTInt& rowsDeleted);
```

ResourceLock object This operation calls DeleteRows in DbInterface object to delete row(s) in RESOURCE_LOCK table. It emulates the Delete SQL statement: 'Delete from RESOURCE_LOCK where "JOB_ID" = this->myJobId'

- GetAttempts

```
EcTInt GetAttempts(void );
```

This operation returns the private attribute that tracks the number of attempts to access the Resource Management database tables.

- GetDelay

```
EcTUInt GetDelay(EcTUInt& cumulative);
```

Lock eligibility flag This operation determines the delay time for lock attempts based on the current value of attribute 'myAttempts'.

- GetState

```
enum DpTPrRlLockState GetState(void );
```

This operation returns the private attribute that tracks the current access state.

- ReadMe

```
EcUtStatus ReadMe();
```

This operation reads the matching object from the database. If the operation is successful, the private attributes are updated. If no records are found, the return detail value is set to one and the routine returns with no further action.

- RelLock

```
EcUtStatus RelLock();
```

This operation set the state of the lock record to 'released', allowing the destructor to free the Resource Management database tables for access by other jobs.

- RWDECLARE_COLLECTABLE

```
int RWDECLARE_COLLECTABLE(DpPrResourceLock );
```

This class is used to implement locking of the Resource Management database tables on a per-job basis. This is necessary in order to maintain the integrity of the information contained there.

- SetAttempts

```
void SetAttempts(EcTInt attempts);
```

This operation sets the private attribute that tracks the number of attempts to access the Resource Management database tables

- SetLock

```
EcUtStatus SetLock();
```

Which subsystem This operation locks the Resource Management database tables for access by the current job. It must be invoked by the DpPrResourceManager constructor. If the tables are already locked, this operation will wait until the current job gets access, unless it times out first.

- SetState

```
void SetState(enum DpTPrRlLockState state);
```

Number of access attempts This operation sets the private attribute that tracks the current access state.

- Sync

```
EcUtStatus Sync();
```

This operation synchronizes the lock object in memory with the values found in the database. If the object exists in the database, it is read in and the attributes are updated. Otherwise, a new object based on current attribute values is written to the database table.

- UpdateLock

```
EcUtStatus UpdateLock(enum DpTPrRlLockState newState);
```

This operation set the state of the lock record to the specified state.

- WriteMe

```
EcUtStatus WriteMe();
```

This operation

7.3.8.3.8 DpPrResourceManager Class

Overview:

This interface class provides an abstract set of operations for effectively managing the collection of processing and storage resources. Proper use of these operations on the part of the Processing System should provide for the same level of resources as were available to the Planning System during plan generation, thereby helping to ensure that what was planned will be processed.

Export Control: Public

Inheritance Relationships:

Attributes:

myDataRoot: RWCString

Partition data files root path

myJobId: DpTPrJobId

Job ID of associated Data Processing Request

myLock: DpPrResourceLock*

Pointer to the database lock record for this object instance

myStringList: RWSlistCollectables

DB Interface String object list

Constructors and Destructor:

DpPrResourceManager(EcUtStatus& rsrcStatus, DpTPrJobId job);

Resource type This constructor will create the object instance for this class and regenerate the resource objects from the persistent database storage. job - The job ID of the associated Data Processing request rsrcStatus - The return status of the constructor operation

~DpPrResourceManager();

This destructor will update the persistent database storage with the current information contained in the resource objects, then effect the deletion of all resource objects before releasing itself.

Operations:

- **AllocateResource**

```
EcUtStatus AllocateResource(RWCString machineIn, RWCString&  
machineOut, EcTInt cpuCount, EcTFloat ramSize);
```

List of FileInfo instances Perform an allocation of CPU and RAM for the amount requested. The allocation is performed for the specified machine and the current job.

- AllocateResource

```
EcUtStatus AllocateResource(RWCString machine, <any> ...);
```

Name of the machine affected Perform an allocation of disk storage for a set of data items and return the set of allocated storage paths,(or perform an allocation for a predetermined set of storage paths ==> Phase 3). The allocation is performed for a the specified machine and the current job.

- BaseName

```
EcTVoid BaseName(RWCString& inPath);
```

File path (input/output) This convenience routine emulates the Unix 'basename' utility. It gets the last token in a slash-separated list. If a trailing '/' character is present, it will be removed first.

- DeallocateResource

```
EcUtStatus DeallocateResource(RWCString machine, <any> ...);
```

Perform a deallocation of disk storage resources for the current job, but only for those storage paths indicated.

- DeallocateResource

```
EcUtStatus DeallocateResource(RWCString machine, EcTInt  
cpuCount, EcTFloat ramSize);
```

List of FileInfo instances Perform a deallocation of processing resources for the amount indicated, for the current job.

- DeallocateResource

```
EcUtStatus DeallocateResource(RWCString machine);
```

Amount of RAM, in bytes, to be deallocated Perform a deallocation of all processing resources for the current job. Processing resources include both CPU and RAM allocations.

- DirName

```
EcTVoid DirName(RWCString& inPath);
```

This convenience routine emulates the Unix 'dirname' utility. It strips off the last token in a slash-separated list. If a trailing '/' character is present, it will be removed first.

- GetAvailableResource

```
EcUtStatus GetAvailableResource(RWCString machine, <any> ...);
```

List of FileInfo instances For a specific machine, retrieve the set of paths which may be allocated for a particular set of data items; no actual allocation has occurred at this point.

- GetBasePath

```
RWCString GetbasePath();
```

Return the base path attribute used in allocation and deallocation operations.

- GetResourceList

```
EcUtStatus GetResourceList(RWSlistCollectables& resourceSet,  
                           enum DpTPrResourceType type);
```

Index to specified element This operation generates a list of all objects of the specified resource type currently found in the resource tables

- GetResource

```
EcUtStatus GetResource(RWCollectable& resource,  
                      RWSlistCollectables& resourceSet, EcTInt element);
```

The resource object Retrieve information about a particular resource object from the list of objects.

- LoadStringSet

```
EcUtStatus LoadStringSet();
```

Job ID of the Data Processing request This routine will create all object instances for the DpPrString class associated with this DpPrResourceManager object, and regenerate the resource objects from the persistent database storage.

- QueryBadResources

```
EcUtStatus QueryBadResources(RWCString machine, <any> ...);
```

TBD Retrieve the set of failed disk allocations for a particular machine.

- QueryResourceState

```
EcUtStatus QueryResourceState(enum DpTPrResourceState& state,  
                            RWCString machine, enum DpTPrResourceType type);
```

List of FileInfo instances Recall resource state for the specified DpPrComputer or DpPrString object

- QueryResourceState

```
EcUtStatus QueryResourceState(enum DpTPrResourceState& state,  
                            RWCString machine, RWCString partition);
```

Resource type to check Recall resource state for the specified DpPrDiskPartition object

- QueryResourceUsage

```
EcUtStatus QueryResourceUsage(RWCString machine, EcTInt& usage);
```

List of FileInfo instances Determine general resource usage by machine

- **QueryResourceUsage**

```
EcUtStatus QueryResourceUsage(EcTInt& usage);
```

TBD Determine general resource usage for the current job.

- **QueryStringSet**

```
EcTVoid QueryStringSet();
```

This routine will step through the list of DpPrString objects associated with this DpPrResourceManager object. Information is output for each element in the list.

- **ReallocateResource**

```
EcUtStatus ReallocateResource(RWCString machine, <any> ...);
```

Amount of RAM requested, in bytes For a specific machine, update the storage allocation values for the specified set of data items.

- **RedeclareResource**

```
EcUtStatus RedeclareResource(RWCString machine, <any> ...);
```

File path (input/output) For a particular machine, redefine the allocation identifiers for the specified set of data items. << NCR ECSed04997

- **ReportResource**

```
EcUtStatus ReportResource(RWCollectable& resource);
```

Partition to check Generate a summary report containing all available information about a particular resource object.

- **UpdateState**

```
EcUtStatus UpdateState(void );
```

Trigger the state update operation for each resource sub-object to get a current assessment of the entire resource pool.

7.3.8.3.9 DpPrResource Class

Overview:

This base class is used to capture the similar features of the derived classes, and to provide for future expansion. All "protected" attributes will be inherited by the derived classes. As well, the "public" operations will be inherited by the derived classes; with few exceptions, these operations will be used as-is, though the derived class may override the GetState() operation if need be.

Export Control: Public

Inheritance Relationships:

Attributes:

myDbAction: enum DpTPrDbCommand

This base class attribute is inherited by the resource subclasses to maintain the next DataBase operation which needs to be performed.

myId: EcTUInt

This base class attribute is inherited by the resource subclasses to uniquely identify object instances.

myName: RWCString

This base class attribute is inherited by the resource subclasses to provide an identifier which is more meaningful in human terms.

myState: enum DpTPrResourceState

This base class attribute is inherited by the resource subclasses to define the last known operating state of each object instance.

ourSequence: EcTUInt

A unique resource ID generator for all of the derived resource classes.

Constructors and Destructor:

Operations:

- **GetDbCommand**

virtual enum DpTPrDbCommand GetDbCommand();

Retrieve the DB Interface attribute for a derived object instance.

- **GetID**

virtual EcTUInt GetID();

Retrieve the ID attribute for a derived object instance.

- **GetName**

virtual RWCString GetName();

Retrieve the Name attribute for a derived object instance.

- **GetSequence**

```
EcTUIInt GetSequence();
```

DbInterface command to use Retrieve the next available id for a new derived resource object.

- **GetState**

```
virtual enum DpTPrResourceState GetState();
```

Retrieve the State attribute for a derived object instance.

- **SetDbCommand**

```
EcTVoid SetDbCommand(enum DpTPrDbCommand dbFlag);
```

Set the DB Interface attribute for a derived object instance.

7.3.8.3.10 DpPrString Class

Overview:

This class is used to represent a "virtual machine". The virtual machine is a collection of one or more physical machines.

Export Control: Public

Inheritance Relationships:

Inherits from [DpPrResource](#)

Attributes:

```
myComputerSet: RWSlistCollectables
```

This pointer attribute references the collection of DpPrComputer resources which are associated with this object instance.

Constructors and Destructor:

```
DpPrString(EcUtStatus& rsrcStatus, DpTPrMachineId name, enum  
DpTPrResourceState state);
```

Resource type This constructor will create an instance of the String object and initialize all of its attributes.

```
DpPrString();
```

Current runtime state This default constructor will be used to create an empty String object, before the DataBase Interface tools are called to transfer String information from the persistent storage area.

```
~DpPrString();
```

Destroy the DpPrString object instance and the associated collection of DpPrComputer resource objects.

Operations:

- AddComputer

```
EcUtStatus AddComputer(DpTPrMachineId machine);
```

Associate a computer resource with a string.

- DebugPrint

```
EcTVoid DebugPrint();
```

This function prints to standard out the values of the object attributes

- DelComputer

```
EcUtStatus DelComputer(DpTPrMachineId machine);
```

New unique machine name Remove association of a computer resource with a string.

- Delete

```
EcUtStatus Delete(EcTInt& rowsDeletedResource);
```

String resource class object This operation calls DeleteRows in DbInterface object to delete row(s) in RSC_STRING table. It emulates the Delete SQL statement: 'Delete from RSC_STRING where "STRING_ID" = this->myId'

- FindComputer

```
EcUtStatus FindComputer(EcTInt cpuCount, EcTFloat ramSize,  
DpPrComputer** computerObjPtr);
```

Pointer to DpPrComputer object found This routine will search the list of DpPrComputer objects associated with this DpPrString object for an element having adequate free processor and RAM resources to support an allocation. If a match is found, a pointer to the first available DpPrComputer object is with adequate resources is returned. Otherwise the pointer is set to zero.

- FindComputer

```
EcUtStatus FindComputer(RWCString computerName, DpPrComputer**  
computerObjPtr);
```

This routine will search the list of DpPrComputer objects associated with this DpPrString object for an element whose name matches computerName. If a match is found, a pointer to the specified DpPrComputer object is returned. Otherwise the pointer is set to zero.

- GetComputerSet

```
RWSlistCollectables GetComputerSet(DpTPrMachineId machine);
```

Name of machine to remove Return the private attribute pointer myComputerSet which points to a list of computer objects

- GetResourceList

```
EcUtStatus GetResourceList(RWSlistCollectables& resourceSet,  
enum DpTPrResourceType type);
```

Name of the virtual machine This operation generates a list of all objects of the specified resource type currently found in the resource tables

- LoadComputerSet

```
EcUtStatus LoadComputerSet();
```

Number of row(s) deleted This routine will create all object instances for the DpPrComputer class associated with this DpPrString object, and regenerate the resource objects from the persistent database storage.

- QueryComputerSet

```
EcTVoid QueryComputerSet();
```

This routine will step through the list of DpPrComputer objects associated with this DpPrString object. Information is output for each element in the list.

- RWDECLARE_COLLECTABLE

```
int RWDECLARE_COLLECTABLE(DpPrString );
```

This class is used to represent a "virtual machine". The virtual machine is a collection of one or more physical machines.

- UpdateState

```
EcUtStatus UpdateState();
```

Pointer to DpPrComputer object found Trigger the state update operation for each resource sub-object to get a current assessment of the entire resource pool.

7.4 AITTL CSCI

7.4.1 CSCI Overview

The purpose of the Algorithm Integration and Test Tools (AITTL) Computer Software Configuration Item (CSCI) is to provide the software tools required to integrate and test (I&T) the science software at the Distributed Active Archive Center (DAAC). The science software will be developed by a Science Computing Facility (SCF), which may be at a different location than the DAAC.

The division of responsibilities between the DAAC and the SCF is generally the following: The SCF is responsible for developing the science software and ensuring that the generated products are scientifically correct. The DAAC is responsible for integrating the science software into the production environment, ensuring that the software will run safely (i.e., will not interfere with the production environment or with other product generation), and running the software in a production mode. The machine on which the science software is developed at the SCF is liable to be of a different class than the machine on which the software is run at the DAAC.

The developer/operator division that is characteristic of the science software lifecycle causes the DAAC I&T personnel to have certain special requirements. The I&T team needs to be able to do the following things:

- Receive a science software delivery from the SCF. The delivery will be made either electronically or on hard media.
- Examine the science software delivery for correctness and completeness. This includes: examining accompanying documentation, verifying that prescribed coding standards have been followed, and running preliminary static and dynamic diagnostic tools to check for potential errors. The delivered files must also be placed under configuration control.
- Compile and link the delivered source files.
- Run test cases. For the most part, these test cases will be supplied by the SCF as part of the delivery. Also supplied by the SCF, for each test case, will be a set of required input files and a corresponding set of output files. Since some of the input files may already reside at the DAAC, the I&T personnel also need the ability to manually stage inputs from the data servers. The DAAC I&T team will re-run each test case and compare their outputs with those supplied by the SCF. Because the SCF and DAAC machines may have different precision, the file comparison utility needs to be more sophisticated than the usual Unix "diff" tool: it needs to be able to screen out differences that are due only to differences in precision. In addition, the ability to examine product metadata is required.
- Diagnose errors. This requires access to: interactive debuggers, screen dump utilities, data visualization tools, and so on.
- Collect resource requirement statistics. This includes: CPU time, memory requirements, disk space requirements, and so on. The collection of such statistics is required both as a "sanity check", to make sure that the measured requirements match the expected values, and also for the PDPS database, which is used by the Planning and Processing systems to execute the science software properly.
- Write reports and maintain the I&T log.

- Write additional ad-hoc test tools.

Satisfaction of these requirements is distributed across several systems. Compilers, linkers, debuggers, and other development and operating system tools will be furnished by the Algorithm Integration and Test Hardware Configuration Item (AITHW) of the Data Processing Subsystem (DPS) of SDPS, since such utilities are so closely wedded to the processing platforms. In addition, some of the standards checking and profiling requirements will probably be satisfied by AITHW as well, since certain of these capabilities will be found in compilers and development environments. The remaining requirements will be satisfied by AITL.

The AITL-supplied tools therefore fall into the following categories:

- Tools to view science software documentation.
- Tools to check compliance of science software to ESDIS-specified coding standards.
- Code analysis tools.
- Data visualization tools.
- HDF file comparison tools.
- Binary file comparison environment.
- Profiling tool.
- Tool to register science software in the processing system database.
- Tools for inserting PGE input files into the IMF Data Server.
- Tools for writing reports and maintaining the I&T logs.
- Tools for checking Process Control Files and for prohibited functions.

Many of the functions for the Algorithm Integration & Test CSCI were developed and delivered for Interim Release 1(IR-1). The Testbed design approach is based on using the IR-1 release as a basis for which to add Release A capabilities. As in IR-1, the Job Scheduler COTS product, AutoSys, can be used to manage the execution of jobs on AITL HW. These jobs represent the execution of a PGE or associated preparation or post-processing jobs. More information is included on AutoSys in Section 4 of this document. Included in the previously provided information is material related to AutoSys' Operations interfaces.

For additional information on science software integration and test procedures, see also: 205-CD-002-001, Science User's Guide and Operations Procedures Handbook for the ECS Project, Part 4, and JU9403V1, Science Software Integration and Test. For information on the ESDIS science software coding standards and guidelines, see: 423-16-01, Data Production Software and Science Computing Facility (SCF) Standards and Guidelines.

7.4.1.1 CSCI Structure

Table 7.4.1.1-1 provides a list of the computer software components in the AITL CSCI.

Note that all software is callable from the UNIX command line. Alternatively, it can be called from the AIT Manager GUI. Command line versions are shown in brackets [], where they differ from the GUI versions.

In implementing custom code, some SDP Toolkit functions are reused, primarily to track AIT

configuration files and manage temporary files.

Table 7.4.1.1-1 AITTL Computer Software Components

CSC	Description
Documentation Viewing Tools	Tools for displaying and/or printing the science software documentation
Standards Checkers	Tools for checking whether science software follows prescribed coding standards
Code Analysis Tools	Tools for checking for code memory leaks, etc.
Data Visualization Tools	Diagnostic tools which display input, output, and intermediate data files as data dumps, plots, and/or images
ECS HDF Visualization Tools	The EOSView CSC is a part of the Client CSCI and has been documented separately. Refer to Section 2 for the specific document number. EOSView is a visualization tools which provides the capability to view ECS-HDF formatted files
HDF File Comparison Tool	Tool for finding differences between two HDF files
Binary File Comparison Environment	Tool for assisting the DAAC user in writing custom code to find differences between two binary files
Profiling Tools	Tools for measuring the resource requirements of the science software
PGE Processing GUI	GUI for executing science software
Data Server Insert	Inserts PGE input files into the IMF Data Server
Update PDPS Database	Registers a PGE with the processing system
Report Generation Tools	Tools for writing miscellaneous reports and for maintaining the integration and test log
SDP Toolkit-Related Tools	Tool to check process control file format; tool to check that no prohibited functions are used

7.4.2 CSCI Context

The context diagram for the AITTL CSCI is shown in Figure 7.4.2-1.

AITTL has interfaces with two other ECS subsystems: the Data Server Subsystem and the Planning Subsystem.

The purpose of the interface with the Data Server is to insert PGE input files (*Data File*). Requests for data transfers (*Data Transfer Request*) and other information (*Data Server Request*) are sent to the Data Server.

The purpose of the interface with Planning is to update the PDPS database, a database containing resource requirements and other information about each product generation executive (PGE) of the science software. This information is needed by the system to plan and execute the PGEs correctly.

and efficiently. Information about PGEs that are already recorded in the database (*PGE Database Information*) are received from Planning. The required information about a PGE (PGE Profile) is sent to Planning to be placed in the database, as are requests to update or get information from the database (*PGE Database Request*).

The primary interface for AITTL is with the operator(s) responsible for integration and test. The operator sends commands (*I&T Tool Command*) to the various tools, supplies the standards checkers with the desired standards (*Standards*) and guidelines (*Guidelines*), supplies the file comparison utility with thresholds (*File Comparison Threshold*) to filter out differences due to precision differences, provides metadata (*Science Software Metadata*) about the science software delivery to the utility that updates the data server with the new science software, and enters information into reports (*Report Information*) and into the integration and test log (*Log Information*). The tools display various results (*I&T Tool Results*), science software documentation (*Displayed Document*), and integration and test reports (*Displayed I&T Report*) for the operator to examine.

Finally, the tools send hardcopy of science software documentation (*Printed Document*), as well as integration and test reports and tool results (*Printed I&T Report*) to the printer.

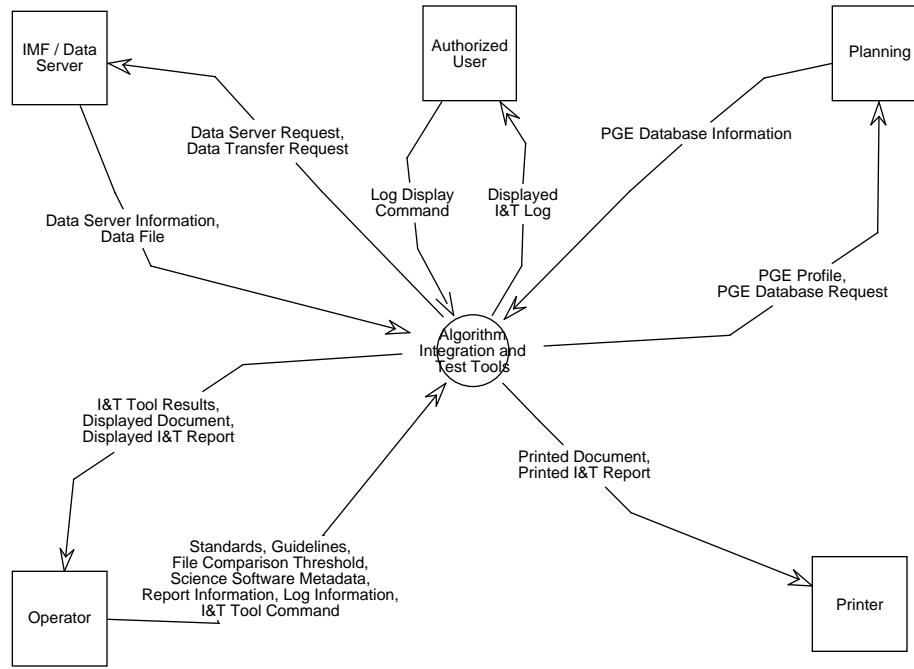


Figure 7.4.2-1. Algorithm Integration and Test Tools Context Diagram

7.4.3 CSCI Object Model

This section gives the object model for the AIT Manager GUI. This GUI is used by the DAAC operator to check in and verify the science software code as delivered by the SCFs. The GUI runs instrument-specific compilation and execution scripts, configuration management scripts, custom code checking, file display and comparison tools, and COTS tools such as office automation and analysis environment programs. The AIT Manager GUI contains a graphical checklist of AI&T steps in delivery and testing of science software, and a display of a log file.

7.4.4 CSCI Functional Model

7.4.4.1 Viewing Science Software Documentation

The context diagram for the tools to display and/or print science software documentation is shown in the data flow diagram, Figure 7.4.4.1-1.

The operator issues a command (*Document Viewing Command*) to view a particular document on the display (*Displayed Document*), or to print a hard copy (*Printed Document*) of a document. Softcopy of the document (*Document*) is stored online in the local integration and test area (*Science Software*).

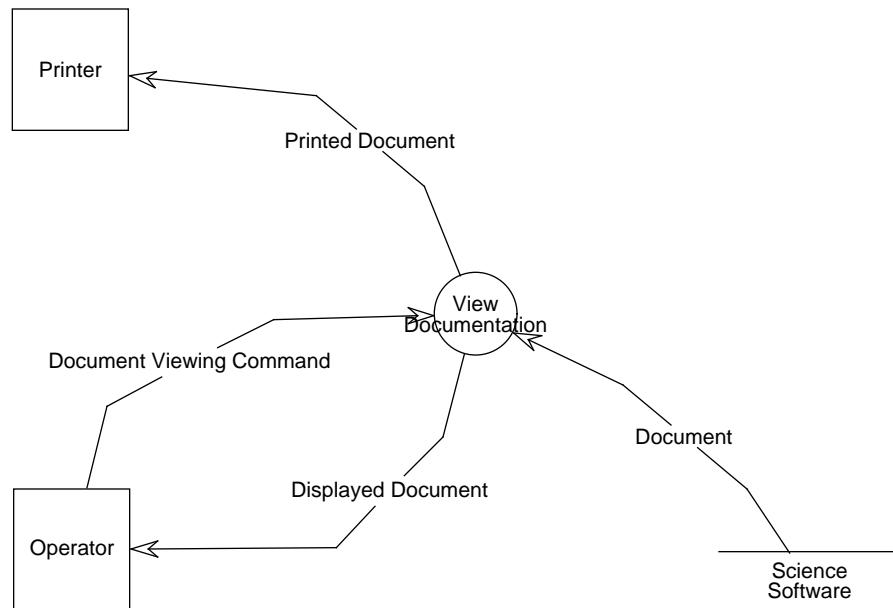


Figure 7.4.4.1-1. View Documentation

7.4.4.2 Checking Coding Standards

The context diagram for the standards checkers is shown in the data flow diagram, Figure 7.4.4.2-1.

The operator issues a command (*Standards Checking Command*) to check for compliance of a particular script (*Shell Script*) or source file (*Source Code*), or set of files, with certain standards and/or guidelines. The operator also must supply the standards checkers with the required standards (*Standards*) and guidelines (*Guidelines*) (only once, when the tools are configured). The results of the check (*Standards Checking Results*) are displayed on the console. Reports may be generated as well, which may be displayed (*Displayed Standards Checking Reports*), printed (*Printed Standards Checking Reports*), and saved as softcopy (*Standards Checking Report*).

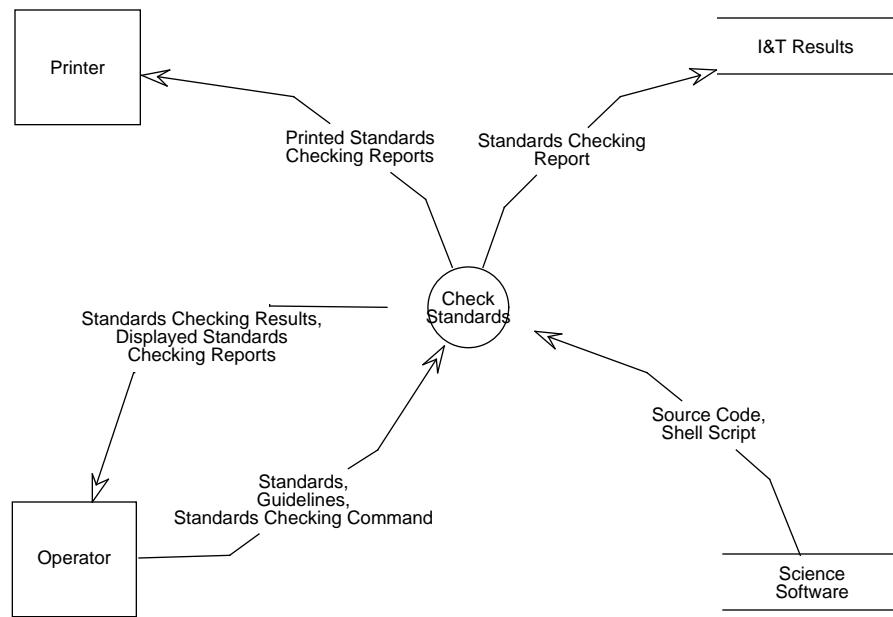


Figure 7.4.4.2-1. Check Standard

7.4.4.3 Analyzing the Code

The context diagram for the static and dynamic code checkers is shown in the data flow diagram, Figure 7.4.4.3-1.

The operator issues a command (*Code Analysis Command*) to make certain checks, either statically on the source files (*Source Code*), or dynamically on the executables (*Executable*). The results (*Code Analysis Results*) of the checks are displayed on the console. Reports may also be generated, and these may be displayed (*Displayed Code Analysis Report*), printed (*Printed Code Analysis Report*), or saved as softcopy (*Code Analysis Report*).

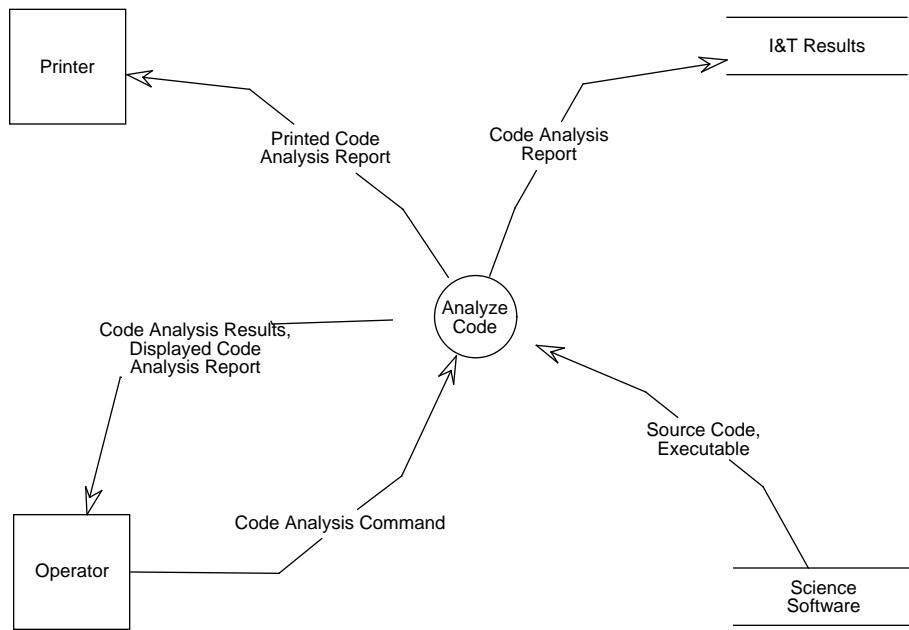


Figure 7.4.4.3-1. Analyze Code

7.4.4.4 Examining the Data

The context diagram for the data visualization tools is shown in the data flow diagram, Figure 7.4.4.4-1.

The operator issues a command (*Data Visualization Command*) to examine a particular data file. The data visualization tools may display the data in the form of a data dump (*Displayed Data Dump*), a two- or three-dimensional plot (*Displayed Plot*), or an image (*Displayed Image*). These displays may also be printed (*Printed Data Dump*, *Printed Plot*, *Printed Image*) or saved as softcopy (*Data Dump*, *Plot*, *Image*).

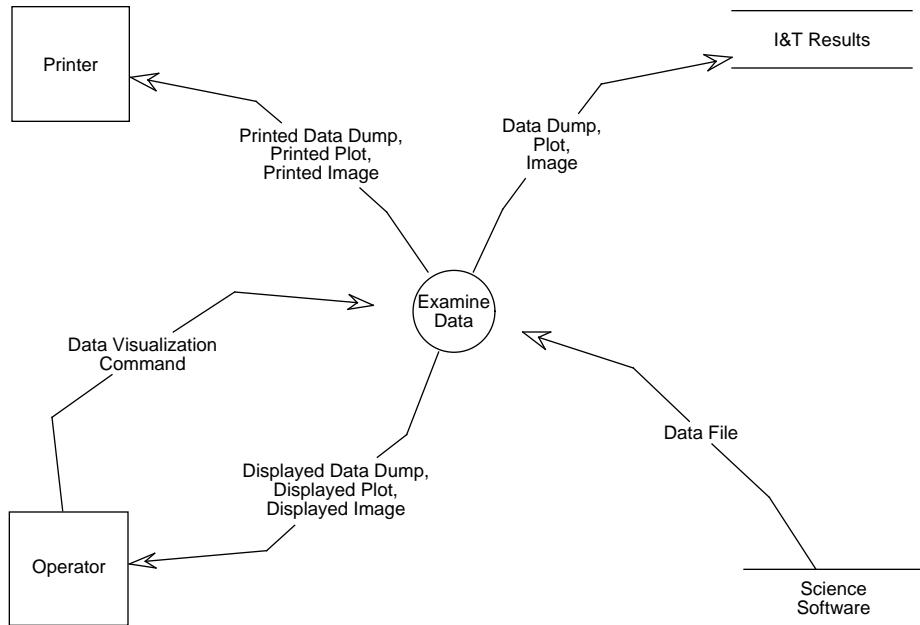


Figure 7.4.4.4-1. Examine Data

7.4.4.5 Comparing Data Files

The context diagram for the file comparison utility is shown in the data flow diagram, Figure 7.4.4.5-1.

The operator issues a command (*File Comparison Command*) to find all differences between two data files (*Data Files to Compare*). Normally, one of these files will have been generated by running a particular test case at the SCF, while the other will have been generated by running the same test case at the DAAC. If there are precision differences between the SCF and DAAC processing platforms, there will be corresponding differences between the two data files which should be ignored. Therefore, the operator will also supply a threshold (*File Comparison Threshold*) for masking out these types of differences. The results (*File Comparison Results*) of the file comparison are displayed on the console. Reports may also be generated, and these may be displayed (*Displayed File Comparison Report*), printed (*Printed File Comparison Report*), or saved as softcopy (*File Comparison Report*). Please note that this section applies directly to HDF files. For binary files, the ECS supplies a programming environment which the DAAC Operator used to write custom binary file comparison code.

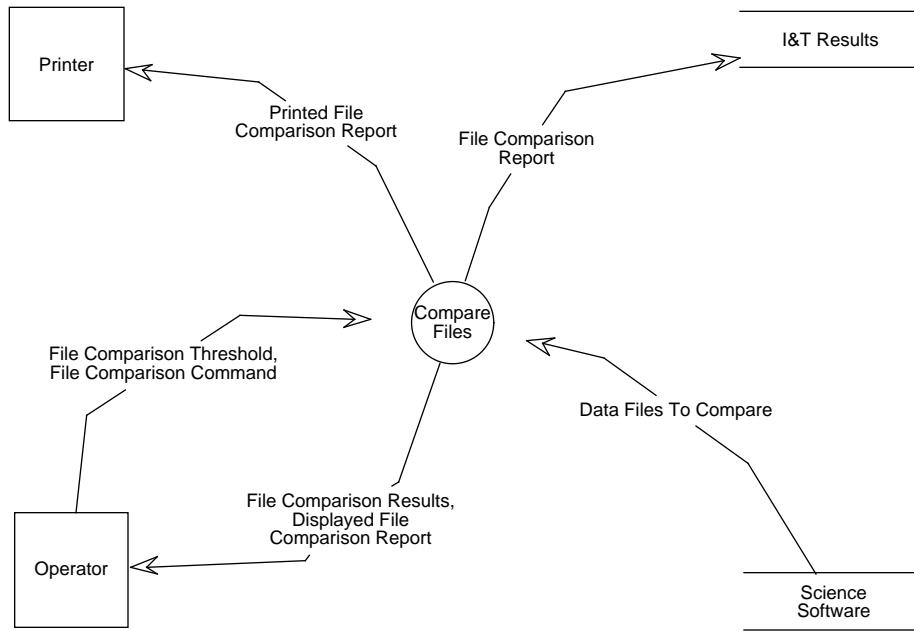


Figure 7.4.4.5-1. Compare Files

7.4.4.6 Measuring Resource Requirements

The context diagram for the profiling tools is shown in the data flow diagram, Figure 7.4.4.6-1.

The operator issues a command (*Profiling Command*) to measure certain resource usage statistics, such as CPU time, memory usage, I/O accesses, disk space requirements, etc., for a specified process or procedure (*Shell Script*, *Executable*). The profiling results (*Profiling Results*) are displayed on the console. Reports may also be generated, and these may be displayed (*Displayed Profiling Report*), printed (*Printed Profiling Report*), or saved as softcopy (*Profiling Report*).

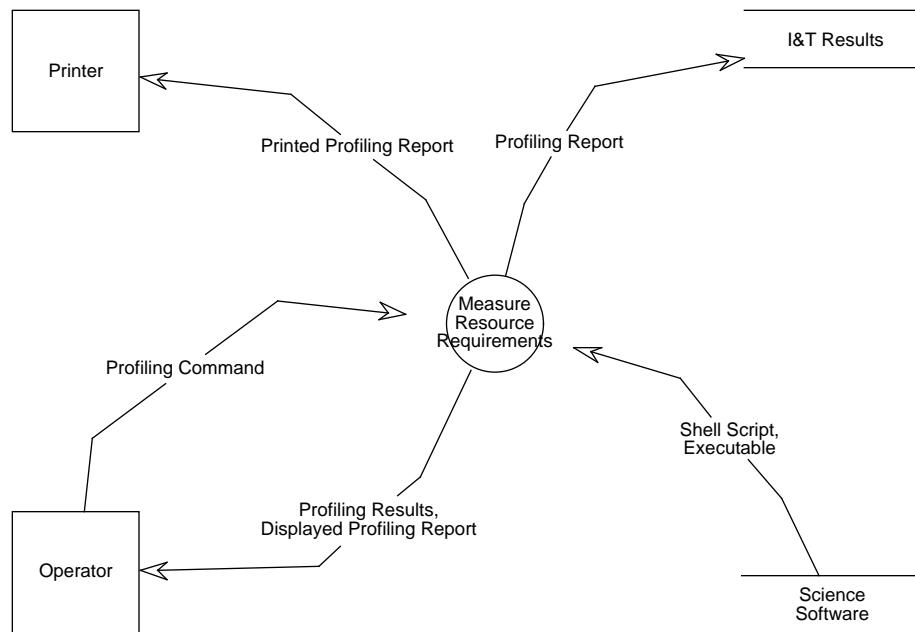


Figure 7.4.4.6-1. Measure Resource Requirements

7.4.4.7 Updating the IMF Data Server

The context diagram for the GUI for archiving the science software files on the IMF data server is shown in the data flow diagram, Figure 7.4.4.7-1.

The purpose of this tool is to allow the user to add a new or updated set of science software files to the IMF data server. This set of files will probably include not only the delivered files (*Science Software*), but also some files generated during integration and test. The operator must also supply metadata (*Science Software Metadata*) for the science software. The operator may issue various commands (*IMF Data Server Update Command*) to the update tool, such as commands to view the current contents of the IMF data server, commands to add the new science software files to the IMF data server, commands to update the metadata, and so on.

Requests (*IMF data server Request*) are sent to the IMF data server to get information about its contents; the IMF data server responds with the requested information (*IMF data server Information*), which is then displayed (*IMF data server Update Display*) on the operator's console. Requests (*IMF data server Request*) to add a new set of science software files are also sent to the IMF data server. The IMF data server responds with the status of the request (*IMF data server Information*), which is also displayed on the console (*IMF data server Update Display*).

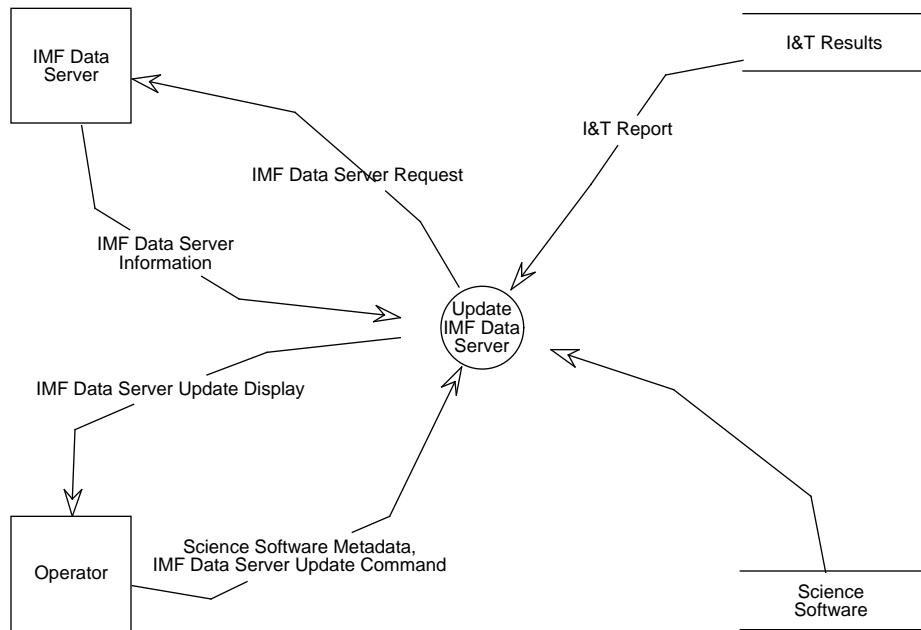


Figure 7.4.4.7-1. Update IMF Data Server

7.4.4.8 Updating the PDPS Database

The context diagram for the GUI for updating the PDPS database is shown in the data flow diagram, Figure 7.4.4.8-1

Information to be loaded into the PDPS database is taken from the profiling reports (*Profiling Report*) (see Section 7.3.6, Measuring Resource Requirements). The operator issues commands (*PDPS Database Update Command*) to add the information to the database (or to view the contents of the database, or add, delete, or modify entries). The PDPS Database is part of an overall Database referred to in the Planning and Processing CSCI design as the PDPS Database. The PDPS Database is a shared database used by Planning, Processing and Algorithm Integration and Test to manage their persistent data. These commands (*PDPS Database Request*) are passed onto the PDPS Database by the GUI, along with the new or modified database entries (*PDPS Installation Package*). Information or status (*PDPS Database Information*, *PDPS Installation Package*) is returned by PDPS Database, and this information (*PDPS Database Update Display*) is displayed on the operator's console.

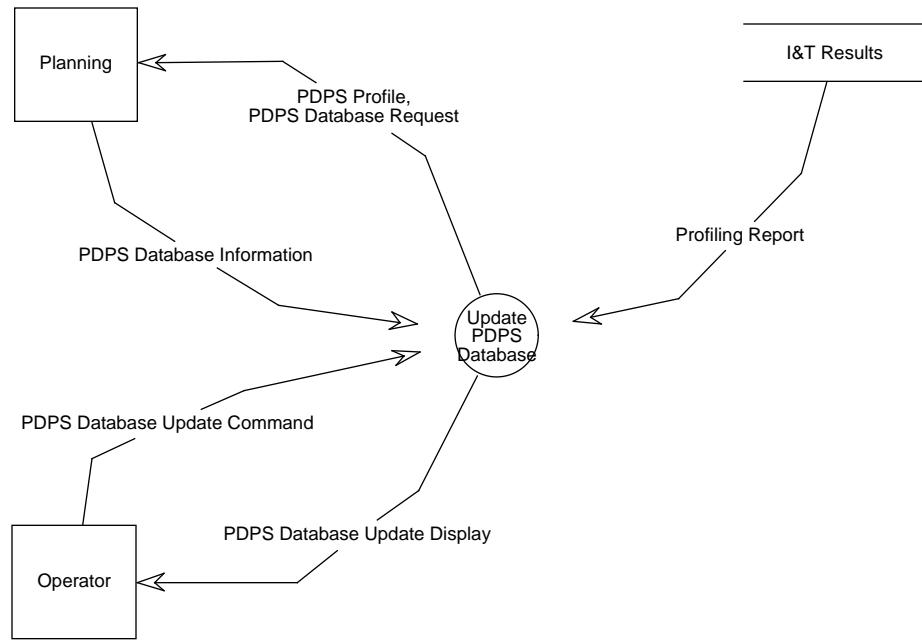


Figure 7.4.4.8-1. Update PDPS Database

7.4.4.9 Writing Reports and Maintaining Logs

The context diagram for tools to generate and maintain the integration and test reports and logs is shown in the data flow diagram, Figure 7.4.4.9-1.

The operator issues commands (*Report Management Command*) to create/modify specified reports (*I&T Report*) and logs (*I&T Log*), supplying the required information (*Report Information*, *Log Information*) and/or using information from other manually- or tool-generated reports (*I&T Report*), and using pre-generated report templates (*I&T Templates*). The operator may also display (*Displayed I&T Report*, *Displayed I&T Log*) or print (*Printed I&T Report*) any report or log, and any authorized user may display the log.

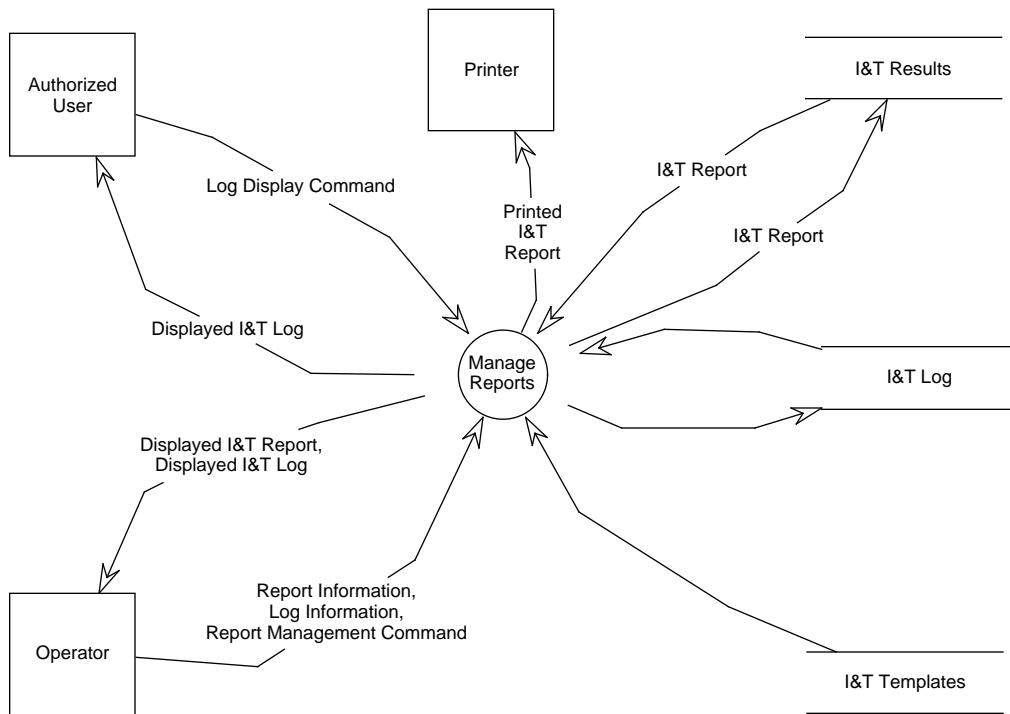


Figure 7.4.4.9-1. Manage Reports

7.4.4.10 AITTL_PreB_Data_Server Class Category

7.4.4.10.1 Overview

DpAtDsrv
myMachineName : RWCString
myFilename : RWCString
myDirname : RWCString
myMdFilename : RWCString
myUr : DsShESDTUR
myEsdtName : RWCString
AcquireFile()
InsertFile()
QueryEsdtNames()
ReadUrFromEmail()
GetEsdtName()
GetMachineName()
GetFilename()
GetDirname()
GetMdFilename()
GetUr()
SetEsdtName()
SetMachineName()
SetDirname()
SetFilename()
SetMdFilename()
SetUr()

7.4.4.10.2

7.4.4.10.2.1

Overview:

This class's methods are the interface between the PDPS/SSIT environment and the Data Server

Export Control:

Inheritance Relationships:

Attributes:

myDirname: RWCString

Name of local directory

myEsdtName: RWCString

Data Server ESDT Short Name

myFilename: RWCString

Name of local file

myMachineName: RWCString

Name of local machine

myMdFilename: RWCString

Name of local ASCII metadata file

myUr: DsShESDTUR

Data Server Universal Reference

Constructors and Destructor:

DpAtDsrv();

Default constructor

**DpAtDsrv(const DsShESDTUR urId, const RWCString filename,
EcUtStatus& status);**

Constructor (used before ACQUIRE)

DpAtDsrv(EcUtStatus& status);

Constructor

**DpAtDsrv(const RWCString esdt, const RWCString filename, const
RWCString mdFilename, EcUtStatus& status);**

Constructor (used before INSERT)

~DpAtDsrv();

Destructor

Operations:

- AcquireFile

EcUtStatus AcquireFile();

Acquires data from the Data Server

- GetDirname

```
const RWCString& GetDirname();
```

- GetEsdtName

```
const RWCString& GetEsdtName();
```

public methods to access persistent attributes

- GetFilename

```
const RWCString& GetFilename();
```

- GetMachineName

```
const RWCString& GetMachineName();
```

- GetMdFilename

```
const RWCString& GetMdFilename();
```

- GetUr

```
const DsShESDTUR& GetUr();
```

- InsertFile

```
EcUtStatus InsertFile();
```

Inserts data to the Data Server

- QueryEsdtNames

```
EcUtStatus QueryEsdtNames(EcTInt* numEsdts, <any> ...);
```

Retrieves all ESDT Short Names from the Data Server

- ReadUrFromEmail

```
EcUtStatus ReadUrFromEmail(const RWCString emailFilename);
```

Reads a UR from an subscription notification email file

- SetDirname

```
EctVoid SetDirname(const RWCString& dirname);
```

- SetEsdtName

```
EcTVoid SetEsdtName(const RWCString& esdtName);
```

- SetFilename

```
EcTVoid SetFilename(const RWCString& filename);
```

- SetMachineName

```
EcTVoid SetMachineName(const RWCString& machineName);
```

- SetMdFilename

```
EcTVoid SetMdFilename(const RWCString& mdFilename);
```

- SetUr

```
EcTVoid SetUr(const DsShESDTUR& ur);
```

7.4.4.11 AITTL_PreB_Metadata_GUI Class Category

7.4.4.11.1 Overview

DpAtOpDbMW
<pre> _dpAtOpDbMW : Widget _dpAtOpDbFORM : Widget _dpAtOpDbTS : Widget _dpAtOpDbSelect : Widget _dpAtOpDbSelPgeVersLIST : Widget _dpAtOpDbSelPgeNameLIST : Widget _dpAtOpDbSelDisPB : Widget _dpAtOpDbSelNewPgeRB : Widget _dpAtOpDbSelExPgeTB : Widget _dpAtOpDbSelMdLBL : Widget _dpAtOpDbSelDelPB : Widget _dpAtOpDbSelHelpPB : Widget _dpAtOpDbProfile : Widget _dpAtOpDbProfCompRB : Widget _dpAtOpDbProfProcStrTB : Widget _dpAtOpDbProfCompStrTB : Widget _dpAtOpDbProfCompStrLIST : Widget _dpAtOpDbProfDiskSpcTF : Widget _dpAtOpDbProfFaultsTF : Widget _dpAtOpDbProfSwapsTF : Widget _dpAtOpDbProfBIOutTF : Widget _dpAtOpDbProfBInputTF : Widget _dpAtOpDbProfMaxMemTF : Widget _dpAtOpDbProfCpuTF : Widget _dpAtOpDbProfWallTimeTF : Widget _dpAtOpDbProfDiskSpcLBL : Widget _dpAtOpDbProfTopSEP : Widget _dpAtOpDbProfMidSEP : Widget _dpAtOpDbProfDiskSpcUnitLBL : Widget _dpAtOpDbProfTitleLBL : Widget _dpAtOpDbProfNumProcSW : Widget _dpAtOpDbProfNumProcLIST : Widget _dpAtOpDbProfNumProcLBL : Widget _dpAtOpDbProfMaxMemUnitLBL : Widget _dpAtOpDbProfCpuSelLBL : Widget _dpAtOpDbProfFaultsLBL : Widget _dpAtOpDbProfSwapsLBL : Widget _dpAtOpDbProfBIOutLBL : Widget _dpAtOpDbProfBInputLBL : Widget _dpAtOpDbProfMaxMemLBL : Widget _dpAtOpDbProfCpuLBL : Widget _dpAtOpDbProfWallSecLBL : Widget _dpAtOpDbProfHelpPB : Widget _dpAtOpDbProfResetPB : Widget _dpAtOpDbProfCommitPB : Widget _dpAtOpDbProfWallLBL : Widget _dpAtOpDbProfPgeNameLBL : Widget _dpAtOpDbProfFRAME : Widget _dpAtOpDbProfPerfStatLBL : Widget _dpAtOpDbProfResReqLBL : Widget _dpAtOpDbRuntime : Widget _dpAtOpDbRunOkPB : Widget _dpAtOpDbRunSelectLIST : Widget _dpAtOpDbRunResetPB : Widget _dpAtOpDbRunSelectLBL : Widget _dpAtOpDbRunTitleLBL : Widget _dpAtOpDbRunPgeNameLBL : Widget _dpAtOpDbRunCommitPB : Widget _dpAtOpDbRunHelpPB : Widget _dpAtOpDbRunDecsLBL : Widget _dpAtOpDbRunDescSW : Widget _dpAtOpDbRunDescTXT : Widget _dpAtOpDbEsdt : Widget _dpAtOpDbEsdtLIST : Widget _dpAtOpDbEsdtPgeNameLBL : Widget _dpAtOpDbEsdtMdDispLBL : Widget _dpAtOpDbEsdtMdLBL : Widget _dpAtOpDbEsdtHelpPB : Widget _dpAtOpDbEsdtLBL : Widget _ecsMenuBar : Widget fileCascade : Widget fileMenu : Widget exit : Widget editCascade : Widget editMenu : Widget undo : Widget cut : Widget copy : Widget paste : Widget clear : Widget viewCascade : Widget viewMenu : Widget commit : Widget helpCascade : Widget helpMenu : Widget onHelp : Widget onContext : Widget onWindow : Widget onKeys : Widget index : Widget tutorial : Widget onVersion : Widget onMode : Widget \$_defaultDpAtOpDbMWResources : String* \$_appDefaults : _UIAppDefault* \$_initAppDefaults : Boolean _clientDataStructs : _UICbStruct*</pre>

DpAtOpDbTextDisMW
<pre> _dpAtOpDbTextDisMW : Widget _dpAtOpDbTextDisFORM : Widget _dpAtOpDbTextDisDonePB : Widget _dpAtOpDbTextDisSW : Widget _dpAtOpDbTextDisTXT : Widget \$_defaultDpAtOpDbTextDisMWResources : String* \$_appDefaults : _UIAppDefault* \$_initAppDefaults : Boolean _clientDataStructs : _UICbStruct* create() className() \$DpAtOpDbTextDisDoneCallback()</pre>

7.4.4.11.2 AITL_PreB_Metadata_GUI Classes

7.4.4.11.2.1 DpAtOpDbMW Class

Overview:

Export Control: [Public](#)

Inheritance Relationships:

Inherits from [UIComponent](#)

Attributes:

`_appDefaults: _UIAppDefault*`

`_clear: Widget`

`_clientDataStructs: _UICbStruct*`

Callback client data.

`_commit: Widget`

`_copy: Widget`

`_cut: Widget`

`_defaultDpAtOpDbMWResources: String*`

Default application and class resources.

`_dpAtOpDbEsdtHelpPB: Widget`

`_dpAtOpDbEsdtLBL: Widget`

`_dpAtOpDbEsdtLIST: Widget`

`_dpAtOpDbEsdtMdDispLBL: Widget`

`_dpAtOpDbEsdtMdLBL: Widget`

`_dpAtOpDbEsdtPgeNameLBL: Widget`

`_dpAtOpDbEsdt: Widget`

`_dpAtOpDbFORM: Widget`

`_DpAtOpDbMW: Widget`

Classes created by this class Widgets created by this class

`_dpAtOpDbProfBlInputLBL: Widget`

`_dpAtOpDbProfBlInputTF: Widget`

`_dpAtOpDbProfBlOutLBL: Widget`

`_dpAtOpDbProfBlOutTF: Widget`

`_dpAtOpDbProfCommitPB: Widget`

`_dpAtOpDbProfCompRB: Widget`

`_dpAtOpDbProfCompStrLIST: Widget`

`_dpAtOpDbProfCompStrTB: Widget`

`_dpAtOpDbProfCpuLBL: Widget`

`_dpAtOpDbProfCpuSecLBL: Widget`

```
_dpAtOpDbProfCpuTF: Widget  
  
_dpAtOpDbProfDiskSpcLBL: Widget  
  
_dpAtOpDbProfDiskSpcTF: Widget  
  
_dpAtOpDbProfDiskSpcUnitLBL: Widget  
  
_dpAtOpDbProfFaultsLBL: Widget  
  
_dpAtOpDbProfFaultsTF: Widget  
  
_dpAtOpDbProffFRAME: Widget  
  
_dpAtOpDbProfHelpPB: Widget  
  
_dpAtOpDbProfile: Widget  
  
_dpAtOpDbProfMaxMemLBL: Widget  
  
_dpAtOpDbProfMaxMemTF: Widget  
  
_dpAtOpDbProfMaxMemUnitLBL: Widget  
  
_dpAtOpDbProfMidSEP: Widget  
  
_dpAtOpDbProfNumProcLBL: Widget  
  
_dpAtOpDbProfNumProcLIST: Widget
```

```
_dpAtOpDbProfNumProcSW: Widget  
  
_dpAtOpDbProfPerfStatLBL: Widget  
  
_dpAtOpDbProfPgeNameLBL: Widget  
  
_dpAtOpDbProfProcStrTB: Widget  
  
_dpAtOpDbProfResetPB: Widget  
  
_dpAtOpDbProfResReqLBL: Widget  
  
_dpAtOpDbProfSwapsLBL: Widget  
  
_dpAtOpDbProfSwapsTF: Widget  
  
_dpAtOpDbProfTitleLBL: Widget  
  
_dpAtOpDbProfTopSEP: Widget  
  
_dpAtOpDbProfWallLBL: Widget  
  
_dpAtOpDbProfWallSecLBL: Widget  
  
_dpAtOpDbProfWallTimeTF: Widget  
  
_dpAtOpDbRunCommitPB: Widget  
  
_dpAtOpDbRunDecsLBL: Widget
```

```
_dpAtOpDbRunDescSW: Widget  
  
_dpAtOpDbRunDescTXT: Widget  
  
_dpAtOpDbRunHelpPB: Widget  
  
_dpAtOpDbRunOkPB: Widget  
  
_dpAtOpDbRunPgeNameLBL: Widget  
  
_dpAtOpDbRunResetPB: Widget  
  
_dpAtOpDbRunSelectLBL: Widget  
  
_dpAtOpDbRunSelectLIST: Widget  
  
_dpAtOpDbRuntime: Widget  
  
_dpAtOpDbRunTitleLBL: Widget  
  
_dpAtOpDbSelDelPB: Widget  
  
_dpAtOpDbSelDisPB: Widget  
  
_dpAtOpDbSelDonePB: Widget  
  
_dpAtOpDbSelect: Widget  
  
_dpAtOpDbSelExPgeTB: Widget
```

```
_dpAtOpDbSelHelpPB: Widget  
  
_dpAtOpDbSelMdLBL: Widget  
  
_dpAtOpDbSelNewPgeRB: Widget  
  
_dpAtOpDbSelNewPgeTB: Widget  
  
_dpAtOpDbSelPgeNameLIST: Widget  
  
_dpAtOpDbSelPgeVersLIST: Widget  
  
_dpAtOpDbTS: Widget  
  
_ecsMenuBar: Widget  
  
_editCascade: Widget  
  
_editMenu: Widget  
  
_exit: Widget  
  
_fileCascade: Widget  
  
_fileMenu: Widget  
  
_helpCascade: Widget  
  
_helpMenu: Widget
```

```
_index: Widget

_initAppDefaults: Boolean

_onContext: Widget

_onHelp: Widget

_onKeys: Widget

_onMode: Widget
Begin user code block <protected>
_onVersion: Widget

_onWindow: Widget

_paste: Widget

_tutorial: Widget

_undo: Widget

_viewCascade: Widget

_viewMenu: Widget
```

Constructors and Destructor:

```
DpAtOpDbMW(const char* , Widget );
```

```
DpAtOpDbMW(const char* );
```

```
virtual ~DpAtOpDbMW();
```

Operations:

- className

```
const char* const className();
```

- create

```
virtual void create(Widget );
```

- dpAtOpDbClearUndoBufferCallback

```
static void dpAtOpDbClearUndoBufferCallback(Widget , XtPointer ,  
XtPointer );
```

- dpAtOpDbClearUndoBuffer

```
virtual void dpAtOpDbClearUndoBuffer(Widget , XtPointer ,  
XtPointer );
```

- DpAtOpDbEsdtCallback

```
static void DpAtOpDbEsdtCallback(Widget , XtPointer , XtPointer  
) ;
```

- DpAtOpDbEsdtDisCallback

```
static void DpAtOpDbEsdtDisCallback(Widget , XtPointer ,  
XtPointer );
```

- DpAtOpDbEsdtDis

```
virtual void DpAtOpDbEsdtDis(Widget , XtPointer , XtPointer );
```

- DpAtOpDbEsdtHelpCallback

- DpAtOpDbEsdtHelp
 - `static void DpAtOpDbEsdtHelpCallback(Widget , XtPointer , XtPointer);`
 - `virtual void DpAtOpDbEsdtHelp(Widget , XtPointer , XtPointer);`
- DpAtOpDbEsdt
 - `virtual void DpAtOpDbEsdt(Widget , XtPointer , XtPointer);`
- dpAtOpDbIntegerOnlyCallback
 - `static void dpAtOpDbIntegerOnlyCallback(Widget , XtPointer , XtPointer);`
- dpAtOpDbIntegerOnly
 - `virtual void dpAtOpDbIntegerOnly(Widget , XtPointer , XtPointer);`
- DpAtOpDbProfCommitCallback
 - `static void DpAtOpDbProfCommitCallback(Widget , XtPointer , XtPointer);`
- DpAtOpDbProfCommit
 - `virtual void DpAtOpDbProfCommit(Widget , XtPointer , XtPointer);`
- DpAtOpDbProfCompProcCallback
 - `static void DpAtOpDbProfCompProcCallback(Widget , XtPointer , XtPointer);`
- DpAtOpDbProfCompProc
 - `virtual void DpAtOpDbProfCompProc(Widget , XtPointer , XtPointer);`

- DpAtOpDbProfHelpCallback

```
static void DpAtOpDbProfHelpCallback(Widget , XtPointer ,  
XtPointer );
```

- DpAtOpDbProfHelp

```
virtual void DpAtOpDbProfHelp(Widget , XtPointer , XtPointer );
```

- DpAtOpDbProfResetCallback

```
static void DpAtOpDbProfResetCallback(Widget , XtPointer ,  
XtPointer );
```

- DpAtOpDbProfReset

```
virtual void DpAtOpDbProfReset(Widget , XtPointer , XtPointer );
```

- dpAtOpDbRealOnlyCallback

```
static void dpAtOpDbRealOnlyCallback(Widget , XtPointer ,  
XtPointer );
```

- dpAtOpDbRealOnly

```
virtual void dpAtOpDbRealOnly(Widget , XtPointer , XtPointer );
```

- DpAtOpDbRunHelpCallback

```
static void DpAtOpDbRunHelpCallback(Widget , XtPointer ,  
XtPointer );
```

- DpAtOpDbRunHelp

```
virtual void DpAtOpDbRunHelp(Widget , XtPointer , XtPointer );
```

- DpAtOpDbRunOkCallback

```
static void DpAtOpDbRunOkCallback(Widget , XtPointer , XtPointer  
) ;
```

- DpAtOpDbRunOk

```
virtual void DpAtOpDbRunOk(Widget , XtPointer , XtPointer );
```

- DpAtOpDbRunResetCallback

```
static void DpAtOpDbRunResetCallback(Widget , XtPointer ,  
XtPointer );
```

- DpAtOpDbRunReset

```
virtual void DpAtOpDbRunReset(Widget , XtPointer , XtPointer );
```

- DpAtOpDbRunSelectCallback

```
static void DpAtOpDbRunSelectCallback(Widget , XtPointer ,  
XtPointer );
```

- DpAtOpDbRunSelect

```
virtual void DpAtOpDbRunSelect(Widget , XtPointer , XtPointer );
```

- DpAtOpDbSelDelCallback

```
static void DpAtOpDbSelDelCallback(Widget , XtPointer ,  
XtPointer );
```

- DpAtOpDbSelDel

```
virtual void DpAtOpDbSelDel(Widget , XtPointer , XtPointer );
```

- DpAtOpDbSelDisCallback

```
static void DpAtOpDbSelDisCallback(Widget , XtPointer ,  
XtPointer );
```

- DpAtOpDbSelDis

```
virtual void DpAtOpDbSelDis(Widget , XtPointer , XtPointer );
```

- DpAtOpDbSelDoneCallback

```
static void DpAtOpDbSelDoneCallback(Widget , XtPointer ,  
XtPointer );
```

- DpAtOpDbSelDone

```
virtual void DpAtOpDbSelDone(Widget , XtPointer , XtPointer );
```

- DpAtOpDbSelExPgeCallback

```
static void DpAtOpDbSelExPgeCallback(Widget , XtPointer ,  
XtPointer );
```

- DpAtOpDbSelExPge

```
virtual void DpAtOpDbSelExPge(Widget , XtPointer , XtPointer );
```

- DpAtOpDbSelHelpCallback

```
static void DpAtOpDbSelHelpCallback(Widget , XtPointer ,  
XtPointer );
```

- DpAtOpDbSelHelp

```
virtual void DpAtOpDbSelHelp(Widget , XtPointer , XtPointer );
```

- DpAtOpDbSelNewPgeCallback

```
static void DpAtOpDbSelNewPgeCallback(Widget , XtPointer ,  
XtPointer );
```

- DpAtOpDbSelNewPge

```
virtual void DpAtOpDbSelNewPge(Widget , XtPointer , XtPointer );
```

- DpAtOpDbSelPgeNameCallback

```
static void DpAtOpDbSelPgeNameCallback(Widget , XtPointer ,  
XtPointer );
```

- DpAtOpDbSelPgeName

```
virtual void DpAtOpDbSelPgeName(Widget , XtPointer , XtPointer );
```

- DpAtOpDbSelPgeVerCallback

```
static void DpAtOpDbSelPgeVerCallback(Widget , XtPointer ,  
XtPointer );
```

Callbacks to interface with Motif.

- DpAtOpDbSelPgeVer

```
virtual void DpAtOpDbSelPgeVer(Widget , XtPointer , XtPointer );
```

These virtual functions are called from the private callbacks or event handlers intended to be overridden in derived classes to define actions

- DpAtOpDbStringChangeCallback

```
static void DpAtOpDbStringChangeCallback(Widget , XtPointer ,  
XtPointer );
```

- DpAtOpDbStringChange

```
virtual void DpAtOpDbStringChange(Widget , XtPointer , XtPointer  
) ;
```

- dpAtOpDbVerifyTextCallback

```
static void dpAtOpDbVerifyTextCallback(Widget , XtPointer ,  
XtPointer );
```

- dpAtOpDbVerifyText

```
virtual void dpAtOpDbVerifyText(Widget , XtPointer , XtPointer );
```

- editMenuClearActivateCallback

```
static void editMenuClearActivateCallback(Widget , XtPointer ,  
XtPointer );
```

- editMenuClearActivate

```
virtual void editMenuClearActivate(Widget , XtPointer ,  
XtPointer );
```
- editMenuCopyActivateCallback

```
static void editMenuCopyActivateCallback(Widget , XtPointer ,  
XtPointer );
```
- editMenuCopyActivate

```
virtual void editMenuCopyActivate(Widget , XtPointer , XtPointer  
);
```
- editMenuCutActivateCallback

```
static void editMenuCutActivateCallback(Widget , XtPointer ,  
XtPointer );
```
- editMenuCutActivate

```
virtual void editMenuCutActivate(Widget , XtPointer , XtPointer  
);
```
- editMenuPasteActivateCallback

```
static void editMenuPasteActivateCallback(Widget , XtPointer ,  
XtPointer );
```
- editMenuPasteActivate

```
virtual void editMenuPasteActivate(Widget , XtPointer ,  
XtPointer );
```
- editMenuUndoActivateCallback

```
static void editMenuUndoActivateCallback(Widget , XtPointer ,  
XtPointer );
```

- editMenuUndoActivate

```
virtual void editMenuUndoActivate(Widget , XtPointer , XtPointer  
);
```

- fileMenuExitActivateCallback

```
static void fileMenuExitActivateCallback(Widget , XtPointer ,  
XtPointer );
```

- fileMenuExitActivate

```
virtual void fileMenuExitActivate(Widget , XtPointer , XtPointer  
);
```

- freeListWidget

```
void freeListWidget(Widget );
```

- helpMenuHelp

```
void helpMenuHelp(EcUtCIHelp* );
```

- helpMenuItemActivateCallback

```
static void helpMenuItemActivateCallback(Widget , XtPointer ,  
XtPointer );
```

- helpMenuItemActivate

```
virtual void helpMenuItemActivate(Widget , XtPointer ,  
XtPointer );
```

- helpModeActivateCallback

```
static void helpModeActivateCallback(Widget , XtPointer ,  
XtPointer );
```

Begin user code block <private>

- helpMenuOnContextActivateCallback

- helpMenuOnContextActivate
 - `static void helpMenuOnContextActivateCallback(Widget , XtPointer , XtPointer);`
 - `virtual void helpMenuOnContextActivate(Widget , XtPointer , XtPointer);`
- helpMenuOnHelpActivateCallback
 - `static void helpMenuOnHelpActivateCallback(Widget , XtPointer , XtPointer);`
 - `virtual void helpMenuOnHelpActivate(Widget , XtPointer , XtPointer);`
- helpMenuOnHelpActivate
 - `static void helpMenuOnHelpActivateCallback(Widget , XtPointer , XtPointer);`
 - `virtual void helpMenuOnHelpActivate(Widget , XtPointer , XtPointer);`
- helpMenuOnKeysActivateCallback
 - `static void helpMenuOnKeysActivateCallback(Widget , XtPointer , XtPointer);`
 - `virtual void helpMenuOnKeysActivate(Widget , XtPointer , XtPointer);`
- helpMenuOnVersionActivateCallback
 - `static void helpMenuOnVersionActivateCallback(Widget , XtPointer , XtPointer);`
 - `virtual void helpMenuOnVersionActivate(Widget , XtPointer , XtPointer);`
- helpMenuOnVersionActivate
 - `static void helpMenuOnVersionActivateCallback(Widget , XtPointer , XtPointer);`
 - `virtual void helpMenuOnVersionActivate(Widget , XtPointer , XtPointer);`
- helpMenuOnWindowActivateCallback

- helpMenuOnWindowActivate


```
static void helpMenuOnWindowActivateCallback(Widget , XtPointer ,
, XtPointer );
```
- helpMenuTutorialActivateCallback


```
virtual void helpMenuTutorialActivateCallback(Widget , XtPointer ,
XtPointer );
```
- helpMenuTutorialActivate


```
virtual void helpMenuTutorialActivate(Widget , XtPointer ,
XtPointer );
```
- setup2


```
void setup2();
```
- setup


```
void setup();
```

Begin user code block <public>
- updateList


```
void updateList();
```

7.4.4.11.2.2 DpAtOpDbTextDisMW Class

Overview:

Export Control: Public

Inheritance Relationships:

Inherits from [UIComponent](#)

Attributes:

```
_appDefaults: _UIAppDefault*
_clientDataStructs: _UICbStruct*
Callback client data.

_defaultDpAtOpDbTextDisMWResources: String*
Default application and class resources.

_dpAtOpDbTextDisDonePB: Widget

_dpAtOpDbTextDisFORM: Widget

_DpAtOpDbTextDisMW: Widget
Classes created by this class Widgets created by this class

_dpAtOpDbTextDisSW: Widget

_dpAtOpDbTextDisTXT: Widget

_initAppDefaults: Boolean
```

Constructors and Destructor:

```
DpAtOpDbTextDisMW(const char* , Widget );
DpAtOpDbTextDisMW(const char* );
virtual ~DpAtOpDbTextDisMW();
```

Operations:

- className

```
const char* const className( );
```

- create

```
virtual void create(Widget );
```

- DpAtOpDbTextDisDoneCallback

```
static void DpAtOpDbTextDisDoneCallback(Widget , XtPointer ,  
XtPointer );
```

Callbacks to interface with Motif.

- DpAtOpDbTextDisDone

```
virtual void DpAtOpDbTextDisDone(Widget , XtPointer , XtPointer  
) ;
```

These virtual functions are called from the private callbacks or event handlers intended to be overridden in derived classes to define actions

7.4.4.12 AITTL_PreB_Metadata Class Category

7.4.4.12.1 Overview

DpAtDatabase	
Makeld() UpdateScience() UpdateOpnl() UpdateOpnlPge()	DpAtScienceMd myId : RWCString myFilename : RWCString myName : RWCString myVersion : RWCString mySswId : RWCString
DpAtOperationalMd mySswId : RWCString	MakePgeFilename() MakeEsdtFilename() PgeFileTemplate() ProcessOdlFile() ProcessOdlObject() ProcessOdlParm() GetId() GetFilename() GetName() GetVersion() SetId() SetFilename() SetName() SetVersion()
RetrieveNewPgeNames() CheckMd() RetrievePgeVersions() GetSswId() SetSswId()	

7.4.4.12.2 AITTL_PreB_Metadata

7.4.4.12.2.1

Overview:

This class's methods update the PDPS/SSIT database with PGE and ESDT operational and science metadata.

Export Control:

Inheritance Relationships:

Attributes:

Constructors and Destructor:

`DpAtDatabase();`

Default constructor

`~DpAtDatabase();`

Destructor

Operations:

- `MakeId`

`EcUtStatus MakeId(EcTBoolean MakeId, RWCString* PgeName,
RWCString* PgeVersion, RWCString* PgeId);`

Makes a PgeId from PGE name and version or extracts name and version from PgeId

- `MakeId`

`EcUtStatus MakeId(EcTBoolean MakeId, RWCString* DataTypeId,
RWCString* EsdtName);`

Makes a DataTypeId from ESDT name or extracts name from DataTypeId

- `UpdateOpnlPge`

`EcUtStatus UpdateOpnlPge(PlPge* pge, PlPerformance* performance,
PlResourceRequirement* resourceRequirement, <any> ...);`

Updates PDPS/SSIT Database with PGE Operational Metadata parameters

- `UpdateOpnl`

`EcUtStatus UpdateOpnl(PlPge* pge, PlPerformance* performance,
PlResourceRequirement* resourceRequirement, <any> ...,
RWCString* badParmVal);`

Updates PDPS/SSIT Database with Operational Metadata parameters (GUI callback)

- `UpdateScience`

`EcUtStatus UpdateScience(const RWCString& pgeId,
PlTimeScheduled* time, DpAtScienceMd* science, <any> ..., <any>
..., <any> ...);`

Updates PDPS/SSIT database with all SCIENCE metadata for a given PGE. (overloaded version for old PGEs)

- UpdateScience

```
EcUtStatus UpdateScience(const RWCString& pgeId, DpAtScienceMd*  
science, <any> ..., <any> ..., <any> ...);
```

Updates PDPS/SSIT database with all SCIENCE metadata for a given PGE. (overloaded version for new PGEs)

7.4.4.12.2.2 DpAtOperationalMd Class

Overview:

This class's methods are callbacks for the PDPS/SSIT Database Update GUI, which updates the database with PGE operational metadata.

Export Control: Public

Inheritance Relationships:

Attributes:

```
mySswId: RWCString
```

PGE software ID

Constructors and Destructor:

```
DpAtOperationalMd();
```

Default constructor

```
~DpAtOperationalMd();
```

Destructor

Operations:

- CheckMd

```
EcUtStatus CheckMd(PlPge* pge, PlPerformance* performance,  
PlResourceRequirement* resourceRequirement, <any> ...,  
RWCString* badParmVal);
```

Checks whether all Operational Metadata parameters have been set by the PDPS/SSIT Database Update GUI

- GetSswId

```

const RWCString& GetSswId();
Public methods to access attributes

• RetrieveNewPgeNames

EcUtStatus RetrieveNewPgeNames(EcTChar* ageFlag, <any> ...,
<any> ..., EcTInt* numNames);
/ Retrieves undeleted PGE names from the database.

• RetrievePgeVersions

EcUtStatus RetrievePgeVersions(RWCString PGName, EcTInt*
numPGEVersions, <any> ...);
Retrieves all PGE version strings for a given PGE name from the SSIT/PDPS database

• SetSswId

EcTVoid SetSswId(const RWCString& id);

```

7.4.4.12.2.3 DpAtScienceMd Class

Overview:

This class is used for access and manipulation of PGE and ESDT SCIENCE metadata, contained in ODL files. Meaning and values of attributes depend on whether the class is used to describe PGE metadata or ESDT metadata.

Some methods are used in PDPS/SSIT Operational Metadata GUI callbacks. Others are used by a separate command-line utility to update the database.

Export Control: **Public**

Inheritance Relationships:

Attributes:

myFilename: RWCString

PGE or ESDT SCIENCE metadata ODL filename

myId: RWCString

PI attribute myPgeId (PGE) or myDataTypeId (ESDT)

myName: RWCString

PGE name or ESDT name

mySswId: RWCString

Science software version (unused for ESDTs)

myVersion: RWCString

PGE version (unused for ESDTs)

Constructors and Destructor:

DpAtScienceMd();

Default constructor

DpAtScienceMd(const RWCString& , EcUtStatus& status);

Alternate constructor (for ESDTs)

DpAtScienceMd(const RWCString& , const RWCString& , EcUtStatus& status);

Alternate constructor (for PGEs)

~DpAtScienceMd();

Destructor

Operations:

- GetFilename

const RWCString& GetFilename();

- GetId

const RWCString& GetId();

public methods to access attributes

- GetName

const RWCString& GetName();

- GetVersion

const RWCString& GetVersion();

- MakeEsdtFilename

EcUtStatus MakeEsdtFilename(EcTBoolean MakeFilename, RWCString* DataTypeId, RWCString* EsdtFilename);

Makes an ESDT SCIENCE metadata filename from DataTypeId or extracts DataTypeId from filename

- MakePgeFilename

```
EcUtStatus MakePgeFilename(EcTBoolean MakeFilename, RWCString*  
PgeId, RWCString* PgeFilename);
```

Makes a PGE ESDT SCIENCE metadata filename from PgeId or extracts PgeId from filename

- PgeFileTemplate

```
EcUtStatus PgeFileTemplate(RWCString PCFName);
```

Creates a PGE SCIENCE metadata ODL file template from a PCF

- ProcessOdlFile

```
EcUtStatus ProcessOdlFile(EcTLongInt& ioFlag, RWCString& pgeId,  
<any> ..., <any> ..., <any> ..., <any> ..., <any> ..., <any> ...);
```

Processes a single ODL file: overloaded version for ESDT files

- ProcessOdlFile

```
EcUtStatus ProcessOdlFile(PlTimeScheduled* timeScheduled, <any>  
..., <any> ..., <any> ..., <any> ..., <any> ..., <any>  
..., <any> ..., <any> ..., <any> ..., <any> ...);
```

Processes a single ODL file: overloaded version for PGE files

- ProcessOdlObject

```
EcUtStatus ProcessOdlObject(AGGREGATE& objectNode, EcTLongInt&  
ioFlag, <any> ..., <any> ..., <any> ..., <any> ..., <any> ...);
```

Processes a single ESDT ODL object: "DATA_TYPE_MASTER"

- ProcessOdlObject

```
EcUtStatus ProcessOdlObject(AGGREGATE& objectNode, <any> ...,  
<any> ..., <any> ..., <any> ...);
```

Processes a single PGE ODL object: "EXIT_MESSAGE"

- ProcessOdlObject

```
EcUtStatus ProcessOdlObject(AGGREGATE& objectNode,  
PlTimeScheduled* time, <any> ..., <any> ..., <any> ...);
```

Processes a single PGE ODL object: "PGE_MASTER"

- ProcessOdlObject

```
EcUtStatus ProcessOdlObject(AGGREGATE& objectNode, <any> ...,
<any> ... , <any> ... , <any> ... , <any> ... ,
PlTimeScheduled* time, enum DpTAtPcfSectionId& sectionId,
RWCString& dataTypeId, EcTLongInt& logicalId, <any> ... , <any>
... , <any> ... );
```

Processes a single PGE ODL object: "PCF_ENTRY"

- ProcessOdlObject

```
EcUtStatus ProcessOdlObject(AGGREGATE& objectNode, <any> ...,
<any> ... , RWCString& dataTypeId, EcTLongInt& logicalId, <any>
... , <any> ... , <any> ... );
```

Processes a single PGE ODL object: "METADATA_CHECKS"

- ProcessOdlParm

```
EcUtStatus ProcessOdlParm(RWCString& parmName, enum
DpTAtOdlDataType requiredType, AGGREGATE& objectNode, <any> ...,
<any> ... , <any> ... , EcTVoid* value);
```

Process a single ODL parameter

- SetFilename

```
EcTVoid SetFilename(const RWCString& filename);
```

- SetId

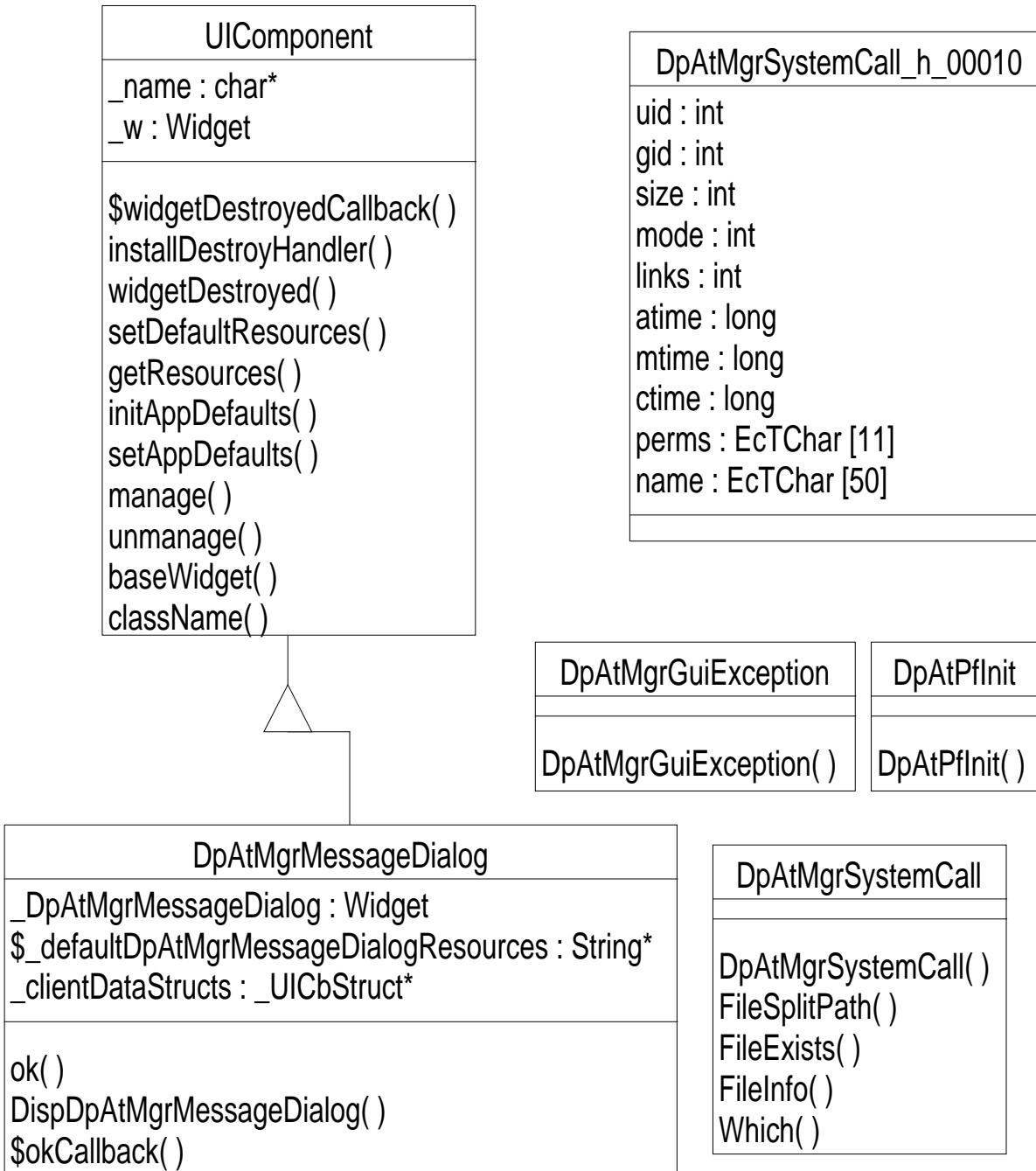
```
EcTVoid SetId(const RWCString& id);
```

- SetName

```
EcTVoid SetName(const RWCString& name);
```

- SetVersion

```
EcTVoid SetVersion(const RWCString& vers);
```



7.4.4.13.2 AITTL_IR1_Heritage_Common_Library Classes

7.4.4.13.2.1 DpAtMgrGuiException Class

Overview:

This class manages GUI message popup windows.

Export Control: **Public**

Inheritance Relationships:

Attributes:

Constructors and Destructor:

DpAtMgrGuiException();

Constructor that pops up a GUI message window

Operations:

7.4.4.13.2.2 DpAtMgrMessageDialog Class

Overview:

Export Control: **Public**

Inheritance Relationships:

Inherits from **UIComponent**

Attributes:

_clientDataStructs: _UICbStruct*

Callback client data.

_defaultDpAtMgrMessageDialogResources: String*

Default class resources

_DpAtMgrMessageDialog: Widget

Widgets created by this class

Constructors and Destructor:

```
DpAtMgrMessageDialog(Widget , enum DpTAtMgrDialogType , EcTChar*  
);
```

```
DpAtMgrMessageDialog(Widget , enum DpTAtMgrDialogType );
```

```
virtual ~DpAtMgrMessageDialog();
```

Operations:

- DispDpAtMgrMessageDialog

```
virtual void DispDpAtMgrMessageDialog(Widget , enum  
DpTAtMgrDialogType );
```

- okCallback

```
static void okCallback(Widget , XtPointer , XtPointer );  
Callbacks to interface with motif
```

- ok

```
virtual void ok(Widget , XtPointer , XtPointer );
```

7.4.4.13.2.3 DpAtMgrSystemCall_h_00010 Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

```
  atime: long  
  
  ctime: long  
  
  gid: int  
  
  links: int  
  
  mode: int  
  
  mtime: long  
  
  name: EcTChar [50]  
  
  perms: EcTChar [11]  
  
  size: int  
  
  uid: int
```

Constructors and Destructor:

Operations:

7.4.4.13.2.4 DpAtMgrSystemCall Class

Overview:

This class manages data for making system calls.

Export Control: **Public**

Inheritance Relationships:

Attributes:

Constructors and Destructor:

`DpAtMgrSystemCall(const EcTChar*);`

Constructor

Operations:

- FileExists

`EcTInt FileExists(EcTChar* name);`

Returns 1 if the file to execute exists

- FileInfo

`EcTInt FileInfo(EcTChar* name, DpAtMgrSystemCall_h_00010* ip);`

Returns info about the file to execute

- FileSplitPath

`EcTInt FileSplitPath(EcTChar* name, EcTChar* path, EcTChar* fname);`

Splits system call file path into directory and file name

- Which

`EcTInt Which(const EcTChar* file);`

Data from unix "which" command

7.4.4.13.2.5 DpAtPfInit Class

Overview:

Export Control: **Public**

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
DpAtPfInit(int argc, char** argv, EcTInt& argc1);  
Constructor
```

Operations:

7.4.4.13.2.6 UIComponent Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

```
_name: char*
```

```
_w: Widget
```

Constructors and Destructor:

```
UIComponent(const char* );  
Protect constructor to prevent direct instantiation  
virtual ~UIComponent();
```

Operations:

- baseWidget

```
const Widget baseWidget();
```

- className

```
virtual const char* const className();
```

Public access functions

- getResources

```
void getResources(const int XtResourceList, const int );
```

Retrieve resources for this class from the resource manager

- initAppDefaults

```
void initAppDefaults(const Widget , const char* , _UIAppDefault* );
```

Initialize the app defaults structure for a class

- installDestroyHandler

```
void installDestroyHandler();
```

- manage

```
virtual void manage();
```

Destructor Manage the entire widget subtree represented by this component. Overrides BasicComponent method

- setAppDefaults

```
void setAppDefaults(const Widget , _UIAppDefault* , const char* inst_name);
```

Set the app defaults for an instance of a class

- setDefaultResources

```
void setDefaultResources(const Widget , const String* );
```

Loads component's default resources into database

- unmanage

```
virtual void unmanage();
```

Manage and unmanage widget tree

- widgetDestroyedCallback

```
static void widgetDestroyedCallback();
```

Interface between XmNdestroyCallback and this class

- widgetDestroyed

```
virtual void widgetDestroyed();
```

Easy hook for derived classes Called by widgetDestroyedCallback() if base widget is destroyed

7.4.4.14 AITTL_IR1_Heritage_File_Differencing_Tools Class Category

7.4.4.14.1 Overview

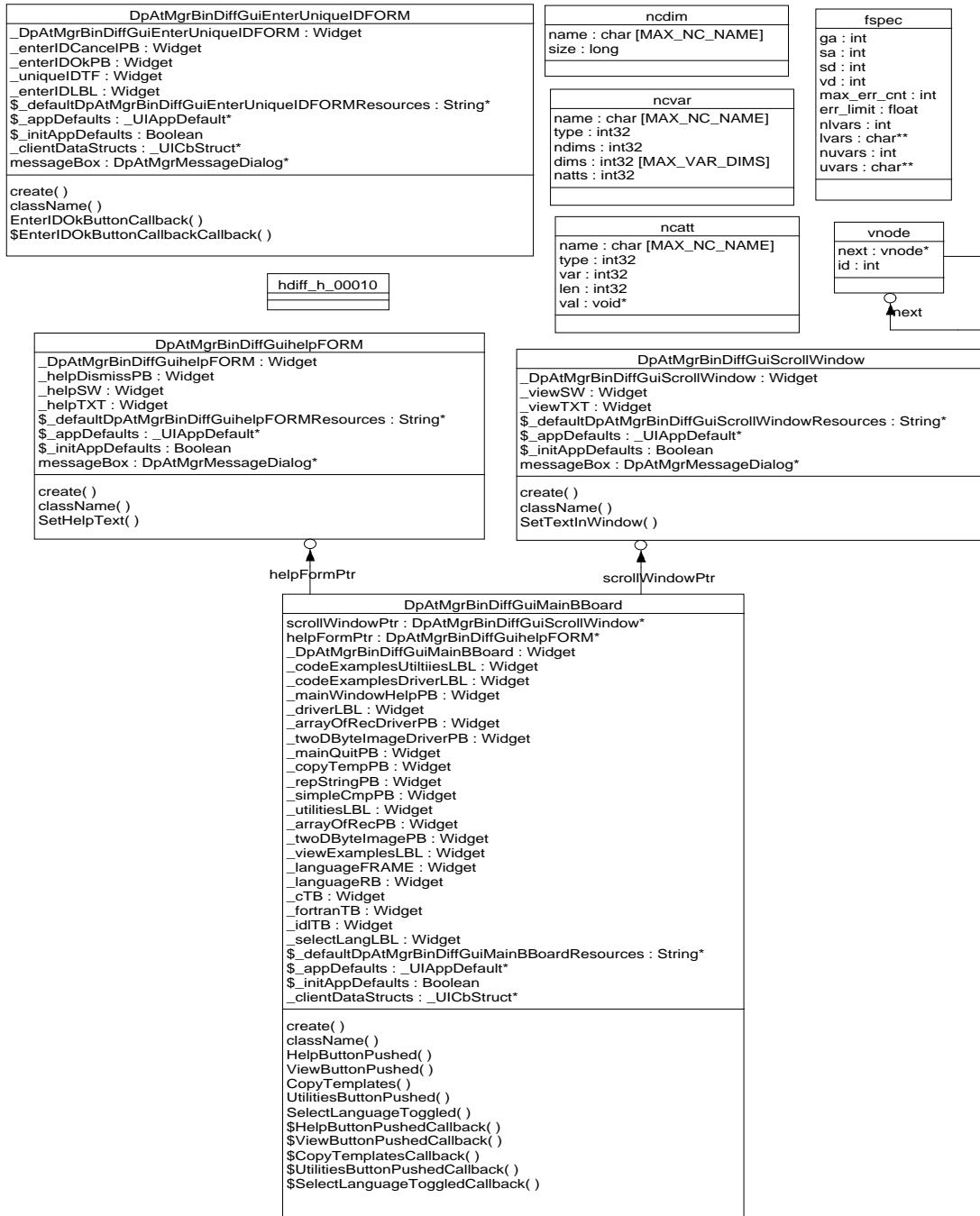


Figure 7.4.4.14.1-1 IR1_Heritage_File_Differencing_Tools

7.4.4.14.2 AITTL_IR1_Heritage_File_Differencing_Tools Classes

7.4.4.14.2.1 DpAtMgrBinDiffGuiEnterUniqueIDFORM Class

Overview:

Export Control: [Public](#)

Inheritance Relationships:

Inherits from [UIComponent](#)

Attributes:

`messageBox: DpAtMgrMessageDialog*`

Begin user code block <private>

`_appDefaults: _UIAppDefault*`

`_clientDataStructs: _UICbStruct*`

Callback client data.

`_defaultDpAtMgrBinDiffGuiEnterUniqueIDFORMResources: String*`

Default application and class resources.

`_DpAtMgrBinDiffGuiEnterUniqueIDFORM: Widget`

Classes created by this class Widgets created by this class

`_enterIDCancelPB: Widget`

`_enterIDLBL: Widget`

`_enterIDOkPB: Widget`

`_initAppDefaults: Boolean`

`_uniqueIDTF: Widget`

Constructors and Destructor:

```
DpAtMgrBinDiffGuiEnterUniqueIDFORM(const char* , Widget );  
  
DpAtMgrBinDiffGuiEnterUniqueIDFORM(const char* );  
  
virtual ~DpAtMgrBinDiffGuiEnterUniqueIDFORM();
```

Operations:

- className

```
const char* const className();
```

- create

```
virtual void create(Widget );
```

- EnterIDOkButtonCallback

```
static void EnterIDOkButtonCallbackCallback(Widget , XtPointer ,  
XtPointer );
```

Callbacks to interface with Motif.

- EnterIDOkButtonCallback

```
virtual void EnterIDOkButtonCallback(Widget , XtPointer ,  
XtPointer );
```

These virtual functions are called from the private callbacks or event handlers intended to be overridden in derived classes to define actions

7.4.4.14.2.2 DpAtMgrBinDiffGuihelpFORM Class

Overview:

Export Control: **Public**

Inheritance Relationships:

Inherits from [UIComponent](#)

Attributes:

`messageBox: DpAtMgrMessageDialog*`

Callbacks to interface with Motif.

Begin user code block <private>

`_appDefaults: _UIAppDefault*`

`_defaultDpAtMgrBinDiffGuihelpFORMResources: String*`

Default application and class resources.

`_DpAtMgrBinDiffGuihelpFORM: Widget`

Classes created by this class Widgets created by this class

`_helpDismissPB: Widget`

`_helpSW: Widget`

`_helpTXT: Widget`

`_initAppDefaults: Boolean`

Constructors and Destructor:

`DpAtMgrBinDiffGuihelpFORM(const char* , Widget);`

`DpAtMgrBinDiffGuihelpFORM(const char*);`

`virtual ~DpAtMgrBinDiffGuihelpFORM();`

Operations:

- `className`

`const char* const className();`

- create

```
virtual void create(Widget );
```

- SetHelpText

```
void SetHelpText(char* );
```

Begin user code block <public>

7.4.4.14.2.3 DpAtMgrBinDiffGuiMainBBoard Class

Overview:

Export Control: Public

Inheritance Relationships:

Inherits from [UIComponent](#)

Attributes:

```
helpFormPtr: DpAtMgrBinDiffGuihelpFORM*
```

Pointer to scroll window class

```
scrollWindowPtr: DpAtMgrBinDiffGuiscrollWindow*
```

Begin user code block <public>

```
_appDefaults: _UIAppDefault*
```

```
_arrayOfRecDriverPB: Widget
```

```
_arrayOfRecPB: Widget
```

```
_clientDataStructs: _UICbStruct*
```

Callback client data.

```
_codeExamplesDriverLBL: Widget
```

```
_codeExamplesUtilitiesLBL: Widget
```

`_copyTempPB: Widget`

`_cTB: Widget`

`_defaultDpAtMgrBinDiffGuiMainBBoardResources: String*`

Default application and class resources.

`_DpAtMgrBinDiffGuiMainBBoard: Widget`

Classes created by this class Widgets created by this class

`_driverLBL: Widget`

`_fortranTB: Widget`

`_idlTB: Widget`

`_initAppDefaults: Boolean`

`_languageFRAME: Widget`

`_languageRB: Widget`

`_mainQuitPB: Widget`

`_mainWindowHelpPB: Widget`

`_repStringPB: Widget`

`_selectLangLBL: Widget`

`_simpleCmpPB: Widget`

```
_twoDByteImageDriverPB: Widget
```

```
_twoDByteImagePB: Widget
```

```
_utilitiesLBL: Widget
```

```
_viewExamplesLBL: Widget
```

Constructors and Destructor:

```
DpAtMgrBinDiffGuiMainBBoard(const char* , Widget );
```

```
DpAtMgrBinDiffGuiMainBBoard(const char* );
```

```
virtual ~DpAtMgrBinDiffGuiMainBBoard();
```

Operations:

- className

```
const char* const className();
```

- CopyTemplatesCallback

```
static void CopyTemplatesCallback(Widget , XtPointer , XtPointer );
```

- CopyTemplates

```
virtual void CopyTemplates(Widget , XtPointer , XtPointer );
```

- create

```
virtual void create(Widget );
```

- HelpButtonPushedCallback

```
static void HelpButtonPushedCallback(Widget , XtPointer ,  
XtPointer );
```

Callbacks to interface with Motif.

- HelpButtonPushed

```
virtual void HelpButtonPushed(Widget , XtPointer , XtPointer );
```

These virtual functions are called from the private callbacks or event handlers intended to be overridden in derived classes to define actions

- SelectLanguageToggledCallback

```
static void SelectLanguageToggledCallback(Widget , XtPointer ,  
XtPointer );
```

- SelectLanguageToggled

```
virtual void SelectLanguageToggled(Widget , XtPointer ,  
XtPointer );
```

- UtilitiesButtonPushedCallback

```
static void UtilitiesButtonPushedCallback(Widget , XtPointer ,  
XtPointer );
```

- UtilitiesButtonPushed

```
virtual void UtilitiesButtonPushed(Widget , XtPointer ,  
XtPointer );
```

- ViewButtonPushedCallback

```
static void ViewButtonPushedCallback(Widget , XtPointer ,  
XtPointer );
```

- ViewButtonPushed

```
virtual void ViewButtonPushed(Widget , XtPointer , XtPointer );
```

7.4.4.14.2.4 DpAtMgrBinDiffGuiScrollWindow Class

Overview:

Export Control: Public

Inheritance Relationships:

Inherits from [UIComponent](#)

Attributes:

`messageBox: DpAtMgrMessageDialog*`

Callbacks to interface with Motif.

Begin user code block <private>

`_appDefaults: _UIAppDefault*`

`_defaultDpAtMgrBinDiffGuiScrollWindowResources: String*`

Default application and class resources.

`_DpAtMgrBinDiffGuiScrollWindow: Widget`

Classes created by this class Widgets created by this class

`_initAppDefaults: Boolean`

`_viewSW: Widget`

`_viewTXT: Widget`

Constructors and Destructor:

`DpAtMgrBinDiffGuiScrollWindow(const char* , Widget);`

`DpAtMgrBinDiffGuiScrollWindow(const char*);`

`virtual ~DpAtMgrBinDiffGuiScrollWindow();`

Operations:

- className

```
const char* const className();
```

- create

```
virtual void create(Widget );
```

- SetTextInWindow

```
void SetTextInWindow(char* );
```

Begin user code block <public>

7.4.4.14.2.5 fspec Class

Overview:

selection for comparison

Export Control: Public

Inheritance Relationships:**Attributes:**

err_limit: float

* max. no of difference to be printed

ga: int

selection for comparison

lvars: char**

* Number of variables specified with -v option * on command line

max_err_cnt: int

* if true, compare Vdata only

nlvars: int

* limit of difference for the comparison

nuvars: int

* list of variable names specified with -v * option on command line

sa: int

* if true, compare global attributes only

sd: int

* if true, compare SD local attributes only

uvars: char**

* Number of variables specified with -u option * on command line

vd: int

* if true, compare SD data only

Constructors and Destructor:

Operations:

7.4.4.14.2.6 hdiff_h_00010 Class

Overview:

Export Control: [Public](#)

Inheritance Relationships:

Attributes:

Constructors and Destructor:

Operations:

7.4.4.14.2.7 ncatt Class

Overview:

attribute

Export Control: Public

Inheritance Relationships:

Attributes:

`len: int32`

`name: char [MAX_NC_NAME]`

`type: int32`

`val: void*`

`var: int32`

attribute

Constructors and Destructor:

Operations:

7.4.4.14.2.8 ncdim Class

Overview:

dimension

Export Control: Public

Inheritance Relationships:

Attributes:

name: char [MAX_NC_NAME]
dimension
size: long

Constructors and Destructor:

Operations:

7.4.4.14.2.9 ncvar Class

Overview:

variable

Export Control: Public

Inheritance Relationships:

Attributes:

dims: int32 [MAX_VAR_DIMS]

name: char [MAX_NC_NAME]
variable
natts: int32

ndims: int32

type: int32

Constructors and Destructor:

Operations:

7.4.4.14.2.10 vnode Class

Overview:

Export Control: [Public](#)

Inheritance Relationships:

Attributes:

`id: int`

`next: vnode*`

Constructors and Destructor:

Operations:

7.4.4.15 AITTL_IR1_Heritage_Process_Control_File_Checker Class Category

7.4.4.15.1 Overview



7.4.4.15.2 AITTL_IR1_Heritage_Process_Control_File_Checker Classes

7.4.4.15.2.1 DpAtMgrCheckPcfInputFORM Class

Overview:

Export Control: **Public**

Inheritance Relationships:

Inherits from **UIComponent**

Attributes:

myPcTempOutputFilename: EcTChar*

Begin user code block <private>

ofPtr: DpAtMgrCheckPcfOutputFORM*

Begin user code block <public>

_appDefaults: _UIAppDefault*

_clientDataStructs: _UICbStruct*

Callback client data.

_defaultDpAtMgrCheckPcfInputFORMResources: String*

Default application and class resources.

_dpAtMgrCheckPcfInputFILE: Widget

_DpAtMgrCheckPcfInputFORM: Widget

Classes created by this class Widgets created by this class

_dpAtMgrCheckPcfInputHelpDLG: Widget

_dpAtMgrCheckPcfInputHelpFORM: Widget

_dpAtMgrCheckPcfInputHelpPB: Widget

_dpAtMgrCheckPcfInputHelpSW: Widget

```
_dpAtMgrCheckPcfInputHelpTXT: Widget
```

```
_initAppDefaults: Boolean
```

Constructors and Destructor:

```
DpAtMgrCheckPcfInputFORM(const char* , Widget );
```

```
DpAtMgrCheckPcfInputFORM(const char* );
```

```
virtual ~DpAtMgrCheckPcfInputFORM( );
```

Operations:

- BxHelpCBCallback

```
static void BxHelpCBCallback(Widget , XtPointer , XtPointer );
```

Callbacks to interface with Motif.

- BxHelpCB

```
virtual void BxHelpCB(Widget , XtPointer , XtPointer );
```

These virtual functions are called from the private callbacks or event handlers intended to be overridden in derived classes to define actions

- CheckPCFCallback

```
static void CheckPCFCallback(Widget , XtPointer , XtPointer );
```

- CheckPCF

```
virtual void CheckPCF(Widget , XtPointer , XtPointer );
```

- className

```
const char* const className();
```

- create

```
virtual void create(Widget );
```

- GetpcTempOutputFilename

```
virtual EcTChar* GetpcTempOutputFilename();
```

7.4.4.15.2.2 DpAtMgrCheckPcfOutputFORM Class

Overview:

Export Control: **Public**

Inheritance Relationships:

Inherits from [UIComponent](#)

Attributes:

```
_appDefaults: _UIAppDefault*
```

```
_clientDataStructs: _UICbStruct*
```

Callback client data.

```
_defaultDpAtMgrCheckPcfOutputFORMResources: String*
```

Default application and class resources.

```
_DpAtMgrCheckPcfOutputFORM: Widget
```

Classes created by this class Widgets created by this class

```
_dpAtMgrCheckPcfOutputHelpDLG: Widget
```

```
_dpAtMgrCheckPcfOutputHelpFORM: Widget
```

```
_dpAtMgrCheckPcfOutputHelpPB: Widget
```

```
_dpAtMgrCheckPcfOutputHelpSW: Widget
```

`_dpAtMgrCheckPcfOutputHelpTXT: Widget`

`_dpAtMgrCheckPcfOutputPB1: Widget`

`_dpAtMgrCheckPcfOutputPB2: Widget`

`_dpAtMgrCheckPcfOutputPB3: Widget`

`_dpAtMgrCheckPcfOutputPB4: Widget`

`_dpAtMgrCheckPcfOutputPB: Widget`

`_dpAtMgrCheckPcfOutputSaveDLG: Widget`

`_dpAtMgrCheckPcfOutputSaveFILE: Widget`

`_dpAtMgrCheckPcfOutputSaveFORM: Widget`

`_dpAtMgrCheckPcfOutputSaveHelpDLG: Widget`

`_dpAtMgrCheckPcfOutputSaveHelpFORM: Widget`

`_dpAtMgrCheckPcfOutputSaveHelpPB: Widget`

`_dpAtMgrCheckPcfOutputSaveHelpSW: Widget`

`_dpAtMgrCheckPcfOutputSaveHelpTXT: Widget`

`_dpAtMgrCheckPcfOutputSEP: Widget`

```
_dpAtMgrCheckPcfOutputSW: Widget
```

```
_dpAtMgrCheckPcfOutputTXT: Widget
```

```
_initAppDefaults: Boolean
```

Constructors and Destructor:

```
DpAtMgrCheckPcfOutputFORM(const char* , Widget );
```

```
DpAtMgrCheckPcfOutputFORM(const char* );
```

```
virtual ~DpAtMgrCheckPcfOutputFORM();
```

Operations:

- className

```
const char* const className();
```

- create

```
virtual void create(Widget );
```

- DispHelpDLG

```
void DispHelpDLG(EcTChar* , Widget , Widget );
```

Begin user code block <public>

- OutputHelpCBCallback

```
static void OutputHelpCBCallback(Widget , XtPointer , XtPointer );
```

Callbacks to interface with Motif.

- OutputHelpCB

```
virtual void OutputHelpCB(Widget , XtPointer , XtPointer );
```

These virtual functions are called from the private callbacks or event handlers intended to be overridden in derived classes to define actions

- PrintFileCallback

```
static void PrintFileCallback(Widget , XtPointer , XtPointer );
```

- PrintFile

```
virtual void PrintFile(Widget , XtPointer , XtPointer );
```

- SaveFileCallback

```
static void SaveFileCallback(Widget , XtPointer , XtPointer );
```

- SaveFile

```
virtual void SaveFile(Widget , XtPointer , XtPointer );
```

- SaveHelpCBCallback

```
static void SaveHelpCBCallback(Widget , XtPointer , XtPointer );
```

- SaveHelpCB

```
virtual void SaveHelpCB(Widget , XtPointer , XtPointer );
```

- SetDefOutputDirCallback

```
static void SetDefOutputDirCallback(Widget , XtPointer ,  
XtPointer );
```

- SetDefOutputDir

```
virtual void SetDefOutputDir(Widget , XtPointer , XtPointer );
```

- SetText

```
void SetText(EcTChar* );
```

7.4.4.16 AITTL_IR1_Heritage_Prohibited_Function_Checker Class Category

7.4.4.16.1 Overview

DpAtMgrProhibitedFunctionGui	
<pre> myProhibitedList : RWCString myCurrentLanguage : enum DpTAtMgrLanguage myBadFunctions : RWTValSortedVector getProhibitedList() findBadFunctions() reset() getToken() isClean() numBadFound() </pre>	<pre> _DpAtMgrProhibitedFunctionGui : Widget _pfCheckerLabelValueLBL : Widget _pfCheckerLBL : Widget _pfCheckerHelpPB : Widget _pfCheckerQuitPB : Widget _pfCheckerRC : Widget _pfCheckerAnalyzePB : Widget _pfCheckerViewPB : Widget _pfCheckerReportPB : Widget _pfCheckerSW : Widget _pfCheckerLIST : Widget _fileSelectorDialog : Widget _fileSelectorFORM : Widget _fileSelectorDirListSW : Widget _fileSelectorDirLIST : Widget _fileSelectorSEP : Widget _fileSelectorDirListLBL : Widget _fileSelectorTextLBL : Widget _fileSelectorFileListLBL : Widget _fileSelectorTextFieldLBL : Widget _fileSelectorTF : Widget _fileSelectorTextSW : Widget _fileSelectorTXT : Widget _fileSelectorHelpPB : Widget _fileSelectorCancelPB : Widget _fileSelectorOkPB : Widget _fileSelectorFileDialog : Widget _sourceCodeFORM : Widget _sourceCodeNumFoundValueLBL : Widget _sourceCodeNumFoundLBL : Widget _sourceCodeLanguageValueLBL : Widget _sourceCodeLanguageLBL : Widget _sourceCodeFileValueLBL : Widget _sourceCodeFileLBL : Widget _sourceCodeLineValueLBL : Widget _sourceCodeLineLBL : Widget _sourceCodeNextPB : Widget _sourceCodeDonePB : Widget _sourceCodeHelpPB : Widget _sourceCodeProhibitedValueLBL : Widget _sourceCodeProhibitedLBL : Widget _sourceCodeSW : Widget _sourceCodeTXT : Widget _helpDialog : Widget _helpFORM : Widget _helpDismissPB : Widget _helpSW : Widget _helpTXT : Widget _reportDialog : Widget _reportFORM : Widget _reportSavePB : Widget _reportPrintPB : Widget _reportCancelPB : Widget _reportHelpPB : Widget _reportSW : Widget _reportTXT : Widget _reportFileDialog : Widget _reportFILE : Widget \$_defaultDpAtMgrProhibitedFunctionGuiResources : String* \$_appDefaults : _UIAppDefault* \$_initAppDefaults : Boolean _clientDataStructs : _UICbStruct* myNextBad : size_t myBadSourceList : RWTValSlist myCurrentDirectory : RWCString myCurrentPosition : size_t oops : DpAtMgrMessageDialog*</pre>

Figure 7.4.4.16.1-1 IR1_Heritage_Prohibited_Function_Checker

7.4.4.16.2 AITTL_IR1_Heritage_Prohibited_Function_Checker Classes

7.4.4.16.2.1 DpAtMgrProhibitedFunctionGui Class

Overview:

Export Control: **Public**

Inheritance Relationships:

Inherits from **UIComponent**

Attributes:

myBadSourceList: RWTValSlist

myCurrentDirectory: RWCString

myCurrentPosition: size_t

myNextBad: size_t

Begin user code block <private>

oops: DpAtMgrMessageDialog*

_appDefaults: _UIAppDefault*

_clientDataStructs: _UICbStruct*

Callback client data.

_defaultDpAtMgrProhibitedFunctionGuiResources: String*

Default application and class resources.

_DpAtMgrProhibitedFunctionGui: Widget

Classes created by this class Widgets created by this class

_fileSelectorCancelPB: Widget

_fileSelectorDialog: Widget

```
_fileSelectorDirListLBL: Widget  
  
_fileSelectorDirListSW: Widget  
  
_fileSelectorDirLIST: Widget  
  
_fileSelectorFileListLBL: Widget  
  
_fileSelectorFileListSW: Widget  
  
_fileSelectorFileLIST: Widget  
  
_fileSelectorFORM: Widget  
  
_fileSelectorHelpPB: Widget  
  
_fileSelectorOkPB: Widget  
  
_fileSelectorSEP: Widget  
  
_fileSelectorTextFieldLBL: Widget  
  
_fileSelectorTextLBL: Widget  
  
_fileSelectorTextSW: Widget  
  
_fileSelectorTF: Widget  
  
_fileSelectorTXT: Widget
```

```
_helpDialog: Widget  
  
_helpDismissPB: Widget  
  
_helpFORM: Widget  
  
_helpSW: Widget  
  
_helpTXT: Widget  
  
_initAppDefaults: Boolean  
  
_pfCheckerAnalyzePB: Widget  
  
_pfCheckerHelpPB: Widget  
  
_pfCheckerLabelValueLBL: Widget  
  
_pfCheckerLBL: Widget  
  
_pfCheckerLIST: Widget  
  
_pfCheckerQuitPB: Widget  
  
_pfCheckerRC: Widget  
  
_pfCheckerReportPB: Widget  
  
_pfCheckersW: Widget
```

```
_pfCheckerViewPB: Widget  
  
_reportCancelPB: Widget  
  
_reportDialog: Widget  
  
_reportFileDialog: Widget  
  
_reportFILE: Widget  
  
_reportFORM: Widget  
  
_reportHelpPB: Widget  
  
_reportPrintPB: Widget  
  
_reportSavePB: Widget  
  
_reportSW: Widget  
  
_reportTXT: Widget  
  
_sourceCodeDialog: Widget  
  
_sourceCodeDonePB: Widget  
  
_sourceCodeFileLBL: Widget  
  
_sourceCodeFileValueLBL: Widget
```

```
_sourceCodeFORM: Widget  
  
_sourceCodeHelpPB: Widget  
  
_sourceCodeLanguageLBL: Widget  
  
_sourceCodeLanguageValueLBL: Widget  
  
_sourceCodeLineLBL: Widget  
  
_sourceCodeLineValueLBL: Widget  
  
_sourceCodeNextPB: Widget  
  
_sourceCodeNumFoundLBL: Widget  
  
_sourceCodeNumFoundValueLBL: Widget  
  
_sourceCodeProhibitedLBL: Widget  
  
_sourceCodeProhibitedValueLBL: Widget  
  
_sourceCodeSW: Widget  
  
_sourceCodeTXT: Widget
```

Constructors and Destructor:

```
DpAtMgrProhibitedFunctionGui(const char* , Widget );
```

```
DpAtMgrProhibitedFunctionGui(const char* );  
  
virtual ~DpAtMgrProhibitedFunctionGui();
```

Operations:

- BxHelpCBCallback

```
static void BxHelpCBCallback(Widget , XtPointer , XtPointer );
```

- BxHelpCB

```
virtual void BxHelpCB(Widget , XtPointer , XtPointer );
```

- ChangeDirectoryDirLISTCallback

```
static void ChangeDirectoryDirLISTCallback(Widget , XtPointer ,  
XtPointer );
```

- ChangeDirectoryDirLIST

```
virtual void ChangeDirectoryDirLIST(Widget , XtPointer ,  
XtPointer );
```

- ChangeDirectoryTFCallback

```
static void ChangeDirectoryTFCallback(Widget , XtPointer ,  
XtPointer );
```

- ChangeDirectoryTF

```
virtual void ChangeDirectoryTF(Widget , XtPointer , XtPointer );
```

- ChangeDirectory

```
void ChangeDirectory(char* );
```

- className

```
const char* const className( );

• create

virtual void create(Widget );

• MyGotSelectionCallback

static void MyGotSelectionCallback(Widget , XtPointer ,
XtPointer );

• MyGotSelection

virtual void MyGotSelection(Widget , XtPointer , XtPointer );

• MyNextCallback

static void MyNextCallback(Widget , XtPointer , XtPointer );

• MyNext

virtual void MyNext(Widget , XtPointer , XtPointer );

• MyUnmanageCallback

static void MyUnmanageCallback(Widget , XtPointer , XtPointer );

• MyUnmanage

virtual void MyUnmanage(Widget , XtPointer , XtPointer );

• PrintReportCallback

static void PrintReportCallback(Widget , XtPointer , XtPointer );

• PrintReport

virtual void PrintReport(Widget , XtPointer , XtPointer );
```

- ProhibitedFuncionHelpCBCallback

```
static void ProhibitedFuncionHelpCBCallback(Widget , XtPointer ,  
XtPointer );
```

Callbacks to interface with Motif.

- ProhibitedFuncionHelpCB

```
virtual void ProhibitedFuncionHelpCB(Widget , XtPointer ,  
XtPointer );
```

These virtual functions are called from the private callbacks or event handlers intended to be overridden in derived classes to define actions

- UpdateSelectionCallback

```
static void UpdateSelectionCallback(Widget , XtPointer ,  
XtPointer );
```

- UpdateSelection

```
virtual void UpdateSelection(Widget , XtPointer , XtPointer );
```

- ViewCallback

```
static void ViewCallback(Widget , XtPointer , XtPointer );
```

- ViewReportCallback

```
static void ViewReportCallback(Widget , XtPointer , XtPointer );
```

- ViewReport

```
virtual void ViewReport(Widget , XtPointer , XtPointer );
```

- View

```
virtual void View(Widget , XtPointer , XtPointer );
```

- WaitCursors

```
void WaitCursors();
```

- WriteReportCallback

```
static void WriteReportCallback(Widget , XtPointer , XtPointer );
```

- WriteReport

```
virtual void WriteReport(Widget , XtPointer , XtPointer );
```

7.4.4.16.2.2 DpAtMgrProhibited Class

Overview:

This class manages prohibited function data.

Export Control: Public

Inheritance Relationships:

Attributes:

myBadFunctions: RWTValSortedVector

list of bad functions found in source

myCurrentLanguage: enum DpTAtMgrLanguage

language for which prohibited function list is valid

myProhibitedList: RWCString

list of prohibited functions

Constructors and Destructor:

DpAtMgrProhibited(const EcTChar* , enum DpTAtMgrLanguage);

language of source code text alternate constructor

DpAtMgrProhibited(const DpAtMgrProhibited&);

language of source code text copy constructor

DpAtMgrProhibited(const RWCString& , enum DpTAtMgrLanguage);

language of source code text alternate constructor

DpAtMgrProhibited();

default constructor

```
DpAtMgrProhibited(enum DpTAtMgrLanguage );
```

alternate constructor

Operations:

- findBadFunctions

```
EcTVoid findBadFunctions(const RWCString* );
```

language to get list for parse input source text for prohibited functions

- getProhibitedList

```
RWBoolean getProhibitedList(enum DpTAtMgrLanguage );
```

get list of prohibited functions for language

- getToken

```
DpTAtMgrToken getToken(size_t ) const;
```

language of source code text get bad function token

- isClean

```
RWBoolean isClean();
```

token number (i.e. list index) returns TRUE if no prohibited functions found, else FALSE

- numBadFound

```
size_t numBadFound() const;
```

returns the total number of prohibited functions found

- operator ==

```
RWBoolean operator ==(const DpAtMgrProhibited& ) const;
```

object whose values are to be assumed defines equality operator

- operator =

```
DpAtMgrProhibited& operator =(const DpAtMgrProhibited& );
```

object used to intialize current object defines assignment operator

- reset

```
EcTVoid reset(const EcTChar* );
```

clear list of prohibited funcs. parse new text

- reset

EcTVoid reset();

object to be tested against clear list of prohibited functions

- reset

EcTVoid reset(const RWCString&);

language of source code text clear list of prohibited funcs., parse new text

- reset

EcTVoid reset(const EcTChar* , enum DpTAtMgrLanguage);

source code text to be parsed clear list of prohibited funcs., parse new text

- reset

EcTVoid reset(const RWCString& , enum DpTAtMgrLanguage);

source code text to be parsed clear list of prohibited funcs., parse new text

7.4.4.17 AITTL_IR1_Heritage_Prolog_Extractor Class Category

7.4.4.17.1 Overview

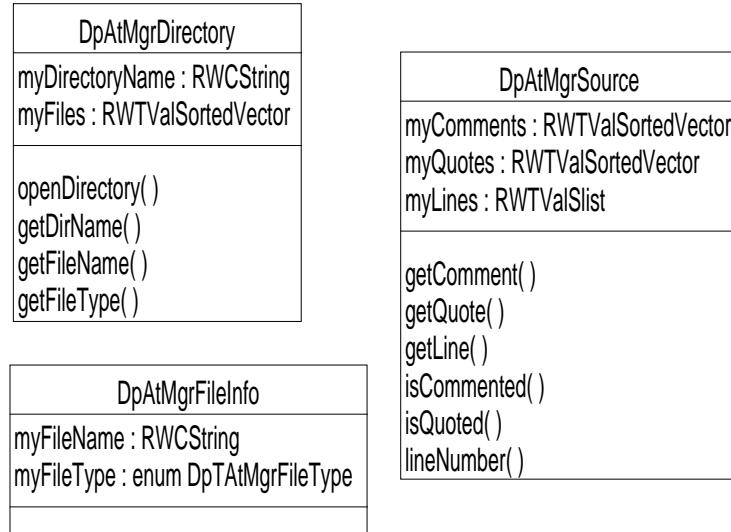


Figure 7.4.4.17.1-1

7.4.4.17.2 AITTL_IR1_Heritage_Prolog_Extractor

7.4.4.17.2.1 DpAtMgrDirectory

Overview:

This class manages directory information. name of directory

Export Control: Public

Inheritance Relationships:

Attributes:

`myDirectoryName`

This class manages directory information. name of directory

`myFiles : RWTValSortedVector`

list of files

Constructors and Destructor:

DpAtMgrDirectory(const EcTChar*);

alternate constructor

Operations:

- **getDirName**

const EcTChar* getDirName();

new directory name return name of current directory

- **getFileName**

const EcTChar* getFileName(size_t);

return a file name

- **getFileType**

enum DpTAtMgrFileType getFileType(size_t);

file list index return a file type

- **getFile**

DpAtMgrFileInfo getFile(size_t);

return a file info structure

- **getNumFiles**

size_t getNumFiles();

directory name return number of files in directory

- **openDirectory**

EcTVoid openDirectory(const EcTChar*);

file list index open a new directory

- **operator =**

EcTVoid operator =(const EcTChar*);

file list index assignment operator

7.4.4.17.2.2 DpAtMgrFileInfo Class

Overview:

Export Control: **Public**

Inheritance Relationships:

Attributes:

myFileName: RWCString

myFileType: enum DpTAtMgrFileType

name of file

Constructors and Destructor:

DpAtMgrFileInfo();

file type (e.g. flat, directory, etc.)

DpAtMgrFileInfo(DpAtMgrFileInfo&);

default constructor

Operations:

- operator <

RWBoolean operator <(const DpAtMgrFileInfo&) const;

structure to be tested against

- operator ==

RWBoolean operator ==(const DpAtMgrFileInfo&) const;

structure whose values are to be assumed

- operator =

DpAtMgrFileInfo& operator =(const DpAtMgrFileInfo&);

structure used to initialize current structure

7.4.4.17.2.3 DpAtMgrSource Class

Overview:

This class holds parsed source code information. list of comment blocks

Export Control: Public

Inheritance Relationships:

Attributes:

myComments: RWTValSortedVector

This class holds parsed source code information. list of comment blocks

myLines: RWTValslist

list of line positions

myQuotes: RWTValSortedVector

list of quoted blocks

Constructors and Destructor:

DpAtMgrSource(const RWCString& , const enum DpTAtMgrLanguage);

language of source code alternate constructor

DpAtMgrSource(const EcTChar* , const enum DpTAtMgrLanguage);

alternate constructor

DpAtMgrSource(const RWCString* , const enum DpTAtMgrLanguage);

language of source code alternate constructor

DpAtMgrSource();

default constructor

Operations:

- **getComment**

RWBoolean getComment(size_t , size_t* , size_t*);

language of source code return comment start and end positions

- **getLine**

```
RWBoolean getLine(size_t , size_t* , size_t* );
```

position of quote end return line number start and end positions

- getQuote

```
RWBoolean getQuote(size_t , size_t* , size_t* );
```

position of comment end return quote start and end positions

- isCommented

```
RWBoolean isCommented(size_t );
```

position of line number end determine if input position is commented

- isQuoted

```
RWBoolean isQuoted(size_t );
```

character position in source code text determine if input position is quoted

- lineNumber

```
size_t lineNumber(size_t );
```

character position in source code text determine line number of input position

- reset

```
EcTVoid reset(const RWCString* , const enum DpTAtMgrLanguage );
```

language of source code clears all previous lists, analyzes text

- reset

```
EcTVoid reset();
```

language of source code clears all lists: myComments, myQuotes, myLines

- reset

```
EcTVoid reset(const EcTChar* , const enum DpTAtMgrLanguage );
```

clears all previous lists, analyzes text

- reset

```
EcTVoid reset(const RWCString& , const enum DpTAtMgrLanguage );
```

language of source code clears all previous lists, analyzes text

7.4.4.18 AITTL_IR1_Heritage_Top_Level_GUI_Manager Class Category

7.4.4.18.1 Overview

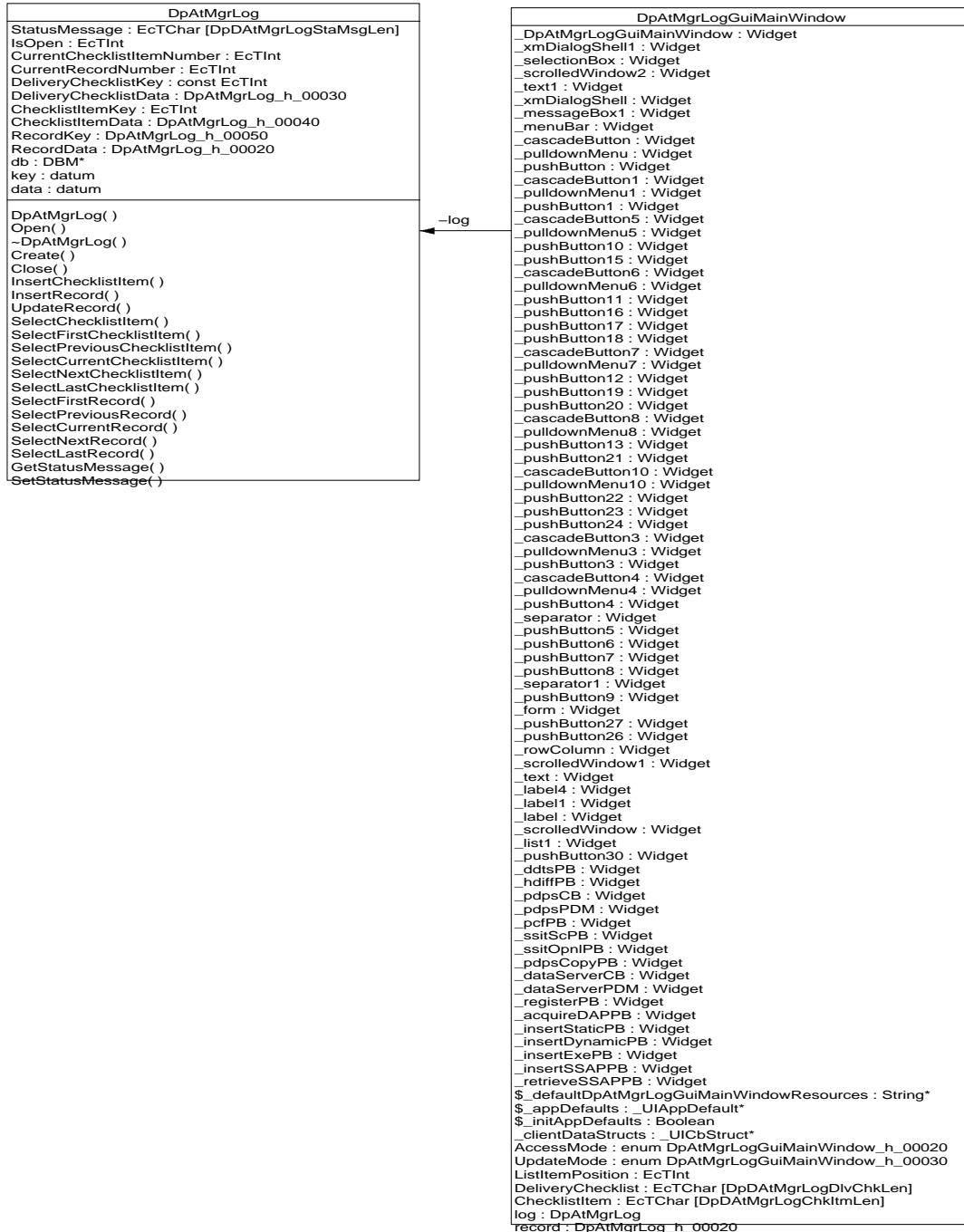


Figure 7.4.4.18.1-1 IR1_Heritage_Top_Level_GUI_Manager

7.4.4.18.2 AITTL_IR1_Heritage_Top_Level_GUI_Manager Classes

7.4.4.18.2.1 DpAtMgrLogGuiMainWindow Class

Overview:

Export Control: **Public**

Inheritance Relationships:

Inherits from **UIComponent**

Attributes:

AccessMode: enum **DpAtMgrLogGuiMainWindow_h_00020**

Private data

ChecklistItem: EcTChar [DpDAtMgrLogChkItmLen]

DeliveryChecklist: EcTChar [DpDAtMgrLogDlvChkLen]

ListWidgetItemPosition: EcTInt

log: **DpAtMgrLog**

record: **DpAtMgrLog_h_00020**

UpdateMode: enum **DpAtMgrLogGuiMainWindow_h_00030**

_acquireDAPPB: Widget

_appDefaults: _UIAppDefault*

_cascadeButton10: Widget

_cascadeButton1: Widget

```
_cascadeButton3: Widget  
  
_cascadeButton4: Widget  
  
_cascadeButton5: Widget  
  
_cascadeButton6: Widget  
  
_cascadeButton7: Widget  
  
_cascadeButton8: Widget  
  
_cascadeButton: Widget  
  
_clientDataStructs: _UICbStruct*  
Callback client data.  
  
_dataServerCB: Widget  
  
_dataServerPDM: Widget  
  
_ddtsPB: Widget  
New widgets added by A.N. Siyyid for Release A  
  
_defaultDpAtMgrLogGuiMainWindowResources: String*  
Default application and class resources.  
  
_DpAtMgrLogGuiMainWindow: Widget  
Classes created by this class Widgets created by this class  
  
_form: Widget  
  
_hdiffPB: Widget
```

```
_initAppDefaults: Boolean  
  
_insertDynamicPB: Widget  
  
_insertExePB: Widget  
  
_insertSSAPPB: Widget  
  
_insertStaticPB: Widget  
  
_label1: Widget  
  
_label4: Widget  
  
_label: Widget  
  
_list1: Widget  
  
_menuBar: Widget  
  
_messageBox1: Widget  
  
_pcfPB: Widget  
  
_pdpsCB: Widget  
  
_pdpsCopyPB: Widget  
  
_pdpsPDM: Widget
```

```
_pulldownMenu10: Widget  
  
_pulldownMenu1: Widget  
  
_pulldownMenu3: Widget  
  
_pulldownMenu4: Widget  
  
_pulldownMenu5: Widget  
  
_pulldownMenu6: Widget  
  
_pulldownMenu7: Widget  
  
_pulldownMenu8: Widget  
  
_pulldownMenu: Widget  
  
_pushButton10: Widget  
  
_pushButton11: Widget  
  
_pushButton12: Widget  
  
_pushButton13: Widget  
  
_pushButton15: Widget  
  
_pushButton16: Widget
```

```
_pushButton17: Widget  
  
_pushButton18: Widget  
  
_pushButton19: Widget  
  
_pushButton1: Widget  
  
_pushButton20: Widget  
  
_pushButton21: Widget  
  
_pushButton22: Widget  
  
_pushButton23: Widget  
  
_pushButton24: Widget  
  
_pushButton26: Widget  
  
_pushButton27: Widget  
  
_pushButton30: Widget  
  
_pushButton3: Widget  
  
_pushButton4: Widget  
  
_pushButton5: Widget
```

```
_pushButton6: Widget  
  
_pushButton7: Widget  
  
_pushButton8: Widget  
  
_pushButton9: Widget  
  
_pushButton: Widget  
  
_registerPB: Widget  
  
_retrieveSSAPPB: Widget  
  
_rowColumn: Widget  
  
_scrolledWindow1: Widget  
  
_scrolledWindow2: Widget  
  
_scrolledWindow: Widget  
  
_selectionBox: Widget  
  
_separator1: Widget  
  
_separator: Widget  
  
_ssitOpn1PB: Widget
```

```
_ssitScPB: Widget  
  
_text1: Widget  
  
_text: Widget  
  
_xmDialogShell1: Widget  
  
_xmDialogShell: Widget
```

Constructors and Destructor:

```
DpAtMgrLogGuiMainWindow(const char* , Widget );  
  
DpAtMgrLogGuiMainWindow(const char* );  
  
virtual ~DpAtMgrLogGuiMainWindow( );
```

Operations:

- className

```
const char* const className();
```

- ConstructItem

```
EcTVoid ConstructItem(EcTChar* buffer);
```

- create

```
virtual void create(Widget );
```

- DynamicMenuCallback

- **DynamicMenuCallback**

```
static void DynamicMenuCallback(Widget , XtPointer , XtPointer );
```

Begin user code block <private>

Private helper functions
- **DynamicMenu**

```
void DynamicMenu(Widget , XtPointer , XtPointer );
```
- **EditLogCallback**

```
static void EditLogCallback(Widget , XtPointer , XtPointer );
```
- **EditLog**

```
virtual void EditLog(Widget , XtPointer , XtPointer );
```
- **EditorCallback**

```
static void EditorCallback(Widget , XtPointer , XtPointer );
```

Callbacks to interface with Motif.
- **Editor**

```
virtual void Editor(Widget , XtPointer , XtPointer );
```

These virtual functions are called from the private callbacks or event handlers intended to be overridden in derived classes to define actions
- **ExternalApplicationCallback**

```
static void ExternalApplicationCallback(Widget , XtPointer , XtPointer ,
```
- **ExternalApplication**

```
virtual void ExternalApplication(Widget , XtPointer , XtPointer );
```
- **GenerateDynamicMenu**

```
virtual void GenerateDynamicMenu(Widget );
```

Begin user code block <protected> Jeff Code VVVVVVV
- **HelpInformationCallback**

- HelpInformation


```
static void HelpInformationCallback(Widget , XtPointer ,
XtPointer );
```

```
virtual void HelpInformation(Widget , XtPointer , XtPointer );
```
- Init


```
EcTVoid Init();
```
- NewLogCallback


```
static void NewLogCallback(Widget , XtPointer , XtPointer );
```
- NewLog


```
virtual void NewLog(Widget , XtPointer , XtPointer );
```
- PostMessage


```
EcTVoid PostMessage(enum DpTAtMgrDialogType dialogtype, EcTChar*
message);
```
- SelectListItemCallback


```
static void SelectListItemCallback(Widget , XtPointer ,
XtPointer );
```
- SelectListItem


```
virtual void SelectListItem(Widget , XtPointer , XtPointer );
```
- StringTrim


```
EcTVoid StringTrim(EcTChar* buffer);
```
- UpdateText


```
EcTVoid UpdateText();
```

7.4.4.18.2.2 DpAtMgrLog Class

Overview:

This class manages SSIT Manager log information.

Export Control: [Public](#)

Inheritance Relationships:

Attributes:

ChecklistItemData: `DpAtMgrLog_h_00040`

Log database structure -- checklist data

ChecklistItemKey: `EcTInt`

Log database structure -- checklist key

CurrentChecklistItemNumber: `EcTInt`

Log database state -- returns current checklist item

CurrentRecordNumber: `EcTInt`

Log database state -- returns current record number

data: `datum`

UNIX ndbm variables -- database data

db: `DBM*`

UNIX ndbm variable -- database name

DeliveryChecklistData: `DpAtMgrLog_h_00030`

Log database structure -- delivery checklist data

DeliveryChecklistKey: `const EcTInt`

Log database structure -- delivery checklist key

IsOpen: `EcTInt`

Log database state -- returns 1 if Open

key: `datum`

UNIX ndbm variables -- database key

RecordData: DpAtMgrLog_h_00020

Log database structure -- record data

RecordKey: DpAtMgrLog_h_00050

Log database structure -- record key

StatusMessage: EcTChar [DpDAtMgrLogStaMsgLen]

Status message buffer

Constructors and Destructor:

DpAtMgrLog();

Constructor

virtual ~DpAtMgrLog();

Destructor

Operations:

- Close

EcTVoid Close();

Log database control interface -- Close

- Create

enum DpAtMgrLog_h_00010 Create(const EcTChar* filename, const EcTChar* DeliveryChecklist);

Log database control interface -- Create

- GetStatusMessage

const EcTChar* GetStatusMessage();

Status message interface

- InsertChecklistItem

enum DpAtMgrLog_h_00010 InsertChecklistItem(const EcTChar* ChecklistItem);

Log database update interface -- Insert a checklist item

- InsertRecord

enum DpAtMgrLog_h_00010 InsertRecord(DpAtMgrLog_h_00020* Record);

Log database update interface -- Insert a record

- Open

```
enum DpAtMgrLog_h_00010 Open(const EcTChar* filename, EcTChar*
DeliveryChecklist);
```

Log database control interface -- Open

- SelectChecklistItem

```
enum DpAtMgrLog_h_00010 SelectChecklistItem(const EcTChar*
ChecklistItem);
```

Checklist Item select interface -- any

- SelectCurrentChecklistItem

```
enum DpAtMgrLog_h_00010 SelectCurrentChecklistItem(EcTChar*
ChecklistItem);
```

Checklist Item select interface -- current

- SelectCurrentRecord

```
enum DpAtMgrLog_h_00010 SelectCurrentRecord(DpAtMgrLog_h_00020*
Record);
```

Record select interface -- current

- SelectFirstChecklistItem

```
enum DpAtMgrLog_h_00010 SelectFirstChecklistItem(EcTChar*
ChecklistItem);
```

Checklist Item select interface -- first

- SelectFirstRecord

```
enum DpAtMgrLog_h_00010 SelectFirstRecord(DpAtMgrLog_h_00020*
Record);
```

Record select interface -- first

- SelectLastChecklistItem

```
enum DpAtMgrLog_h_00010 SelectLastChecklistItem(EcTChar*
ChecklistItem);
```

Checklist Item select interface -- last

- SelectLastRecord

```
enum DpAtMgrLog_h_00010 SelectLastRecord(DpAtMgrLog_h_00020*
Record);
```

Record select interface -- last

- SelectNextChecklistItem

```
enum DpAtMgrLog_h_00010 SelectNextChecklistItem(EcTChar*
ChecklistItem);
```

Checklist Item select interface -- next

- SelectNextRecord

```
enum DpAtMgrLog_h_00010 SelectNextRecord(DpAtMgrLog_h_00020*
Record);
```

Record select interface -- next

- SelectPreviousChecklistItem

```
enum DpAtMgrLog_h_00010 SelectPreviousChecklistItem(EcTChar*
ChecklistItem);
```

Checklist Item select interface -- previous

- SelectPreviousRecord

```
enum DpAtMgrLog_h_00010
SelectPreviousRecord(DpAtMgrLog_h_00020* Record);
```

Record select interface -- previous

- SetStatusMessage

```
EcTVoid SetStatusMessage(const EcTChar* message);
```

Status message helper function

- UpdateRecord

```
enum DpAtMgrLog_h_00010 UpdateRecord(DpAtMgrLog_h_00020*
Record);
```

Log database update interface -- Update a record

8. Integrated Metastorage Factory (IMF)

8.1 Introduction

The Integrated Metastorage Factory (IMF) provides the equivalent level of functionality, required by the Release A PDPS, that the Release A Data Server was intended to perform. Originally developed to support internal PLS/DPS integration testing and implemented in the “C” language, the IMF now serves as the “Data Server” for the Pre-Release B Testbed.

8.2 Overview

As the name suggests, the IMF is fully integrated with the Release A Data Server APIs required for Planning and Data Processing. This feature allows the IMF to be replaced with an alternate Data Server, to the degree that the same API signatures and results-lists formats are employed. In the case of the Testbed, substitution is as easy as redirecting the link statements which are used to construct the Testbed applications. Unlike the Release A Data Server, which was to use EMASS for storage and Sybase for metadata attribute searches, the IMF makes good use of the hierarchical nature of the UNIX file system to store product data on RAID; all of the necessary metadata searches are performed directly on the MCF files which reside within the same directory as the products which they describe. Though less complex than the Release A Data Server approach, this integral storage mechanism for the archival of products and metadata is no less effective.

The actual file structure for the IMF is described in Section 8.4, which further facilitates several aspects of Testbed operations. The following is a list of IMF key characteristics and it’s operations:

1. Flexibility - The entire IMF archive is located beneath a single directory which can be installed anywhere and remotely mounted for convenience (As size requirements for this archive can be significant, use of a large (~100 Gb) RAID partition is preferable).
2. Maintenance - Since products are governed by their data type (ESDT), a distinct directory branch exists (is created) for each data type below which all data products and their metadata reside. The ESDT itself is defined by a hidden descriptor file (i.e., “descriptor”) within each of these directories.
3. Performance - Since metadata searches (e.g., inquiries) are performed on one data type at a time, searches are confined to the metadata files that currently reside in a single directory. Also, unlike, the Release A Data Server which was to perform “ftp” transfers of data from the storage device to the science computing platform, the IMF uses NFS copy to push data from the archive to the local computing host.
4. Expandability - The IMF easily accommodates new data types by introducing additional directory branches within the archive. Once an ESDT is created in the IMF, all of the services required for that ESDT are immediately available (e.g., archive insert). Once again, unlike the Release A Data Server which relied on maintenance intensive dynamic libraries to fulfill service requirements (e.g., DLL), this is achieved, in the IMF, without the need for additional software. This ease of expansion gives the IMF its factory-like characteristics.

8.3 IMF Context

Figure 8.3-1 illustrates the IMF interfaces with the Testbed CIs.

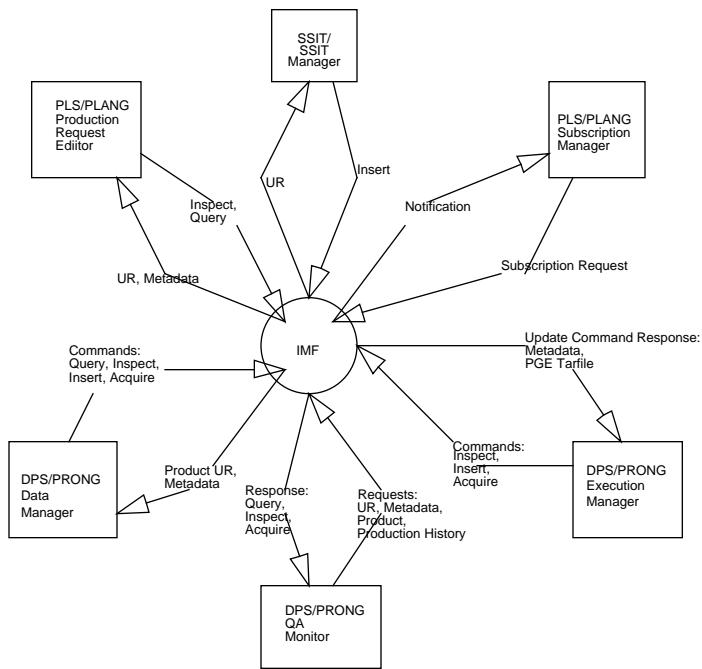


Figure 8.3-1. IMS Context Diagram

8.4 IMF File Structure and Description

The IMF archive itself is defined by a series of UNIX directory branches, each of which completely contains the description of a single ESDT and its granule members. The ESDT itself is described by a single file named “.descriptor”. This file contains metadata attribute information, in an ODL format, that completely defines the set of metadata required to ensure the validity of a member granule. Only granules which satisfy this requirement can become members of the archive. The granules themselves are UNIX files, defined in various formats (e.g., HDF-EOS), which are generated by the science PGEs. Figure 8.4-1 illustrates a structure of the IMF archive. To ensure successful inserts of these granules, the PGEs must also generate metadata files which agree (i.e., have appropriate attribute values) with the ESDT descriptor for that data type. The actual validation of granule metadata is performed by the PDPS during a call to the IMF insert service

(see Section 8.5.1).

The archive can expand, as necessary, to accommodate new ESDTs as PGEs which require them become integrated into the Pre-Release B Testbed (this procedure is performed by SSIT applications during the algorithm integration period). The number of granule inserts is limited only by the number of file entries that UNIX can support for a particular directory.

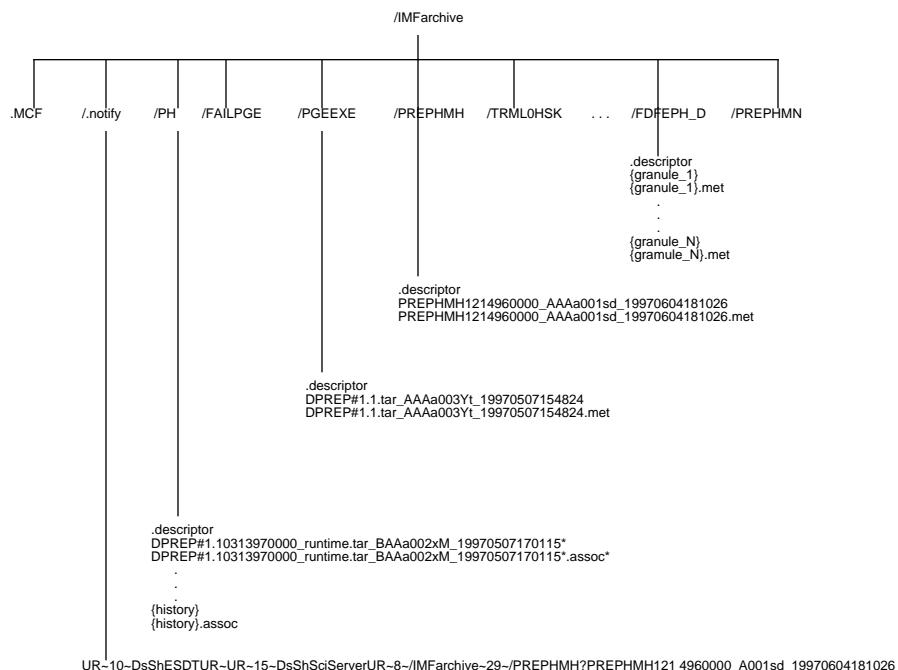


Figure 8.4-1. Structure of IMF Archive

The following is an explanation of some of the standard archive entries, which are shown in Figure 8.4-1.

.descriptor - This file embodies the essence of an ESDT by defining all of the inventory metadata that can be associated with a particular data type.

{granule} - A file which contains the standard product data for a particular ESDT; typically in HDF-EOS format.

{granule}.met - A file which contains the metadata describing the {granule} file of the same name.

{history} - A tar file which contains the production history for a single DPR run.

{history}.assoc - A file which contains universal references for each of the {granules} that are associated with the production history file of the same name.

.MCF - This is the master MCF, used by the SSIT software as the data dictionary reference defining metadata as per the Release A data model.

.notify - This directory contains pseudo-UR entries which are used by the Subscription Manager to forward “insert” information on to the PDPS database.

/PH - This ESDT directory serves as a container for all production history tar file inserts.

/FAILPGE - This ESDT directory serves as a container for all failed PGE diagnostic tar file inserts.

/PGEEEXE - This ESDT directory serves as a container for all PGE executable tar files inserts.

The following represent actual ESDT directories for some of the archive entries associated with data preprocessing for the TRMM platform (a.k.a. DPREP):

/PREPHMH - Preprocessed TRMM level-0 data in HDF format.

/TRML0HSK - TRMM level-0 engineering housekeeping data.

/FDFEPH_D - FDF generated ephemeris data.

/PREPHMN - Preprocessed TRMM level-0 data in native machine format.

8.5 IMF / PDPS Interfaces

Since the PDPS only requires an interface to a subset of the functionality defined by the Science Data Server, only those interfaces were developed within the IMF. Figure 8.5-1 illustrates the IMF/PDPS interfaces.

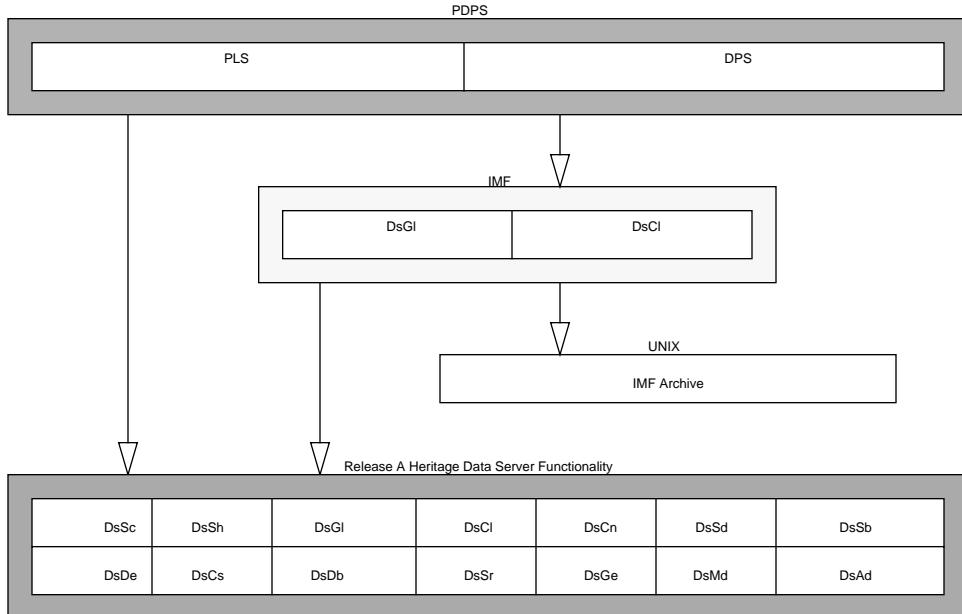


Figure 8.5-1. IMF/PDPS Interface Diagram

The following sub-sections provides a description of those interfaces and the PDPS applications which use these services.

8.5.1 Insert Interfaces

Insert interfaces are used by:

SSIT Manager

Data Manager

Execution Manager

This set of interfaces provides the essential services for the “destaging” of files from the local science platform to the location of the IMF archive. Various types of files, or collections of files, can be inserted, but all such inserts are performed on an ESDT basis. That is, only files belonging to a specific data type can be inserted at any one time. As receipt to acknowledge the successful insert of data, a single universal reference is returned to the caller. This handle may then be used in subsequent calls to retrieve the data from the archive, should that ever become necessary.

- Science Granule Insert - This is the standard interface used within PDPS to archive all science data granules. All such inserts are performed a granule at a time to ensure that a unique UR is received for each granule that is archived.

Coding example for inserting science granule(s):

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <strstream.h>
#include "EcPfClient.h"
#include "DsShSciServerUR.h"
#include "DsCIESDTReferenceCollector.h"
#include "DsCIRequest.h"
#include "DsCIInsertCommand.h"
#include "GIClient.h"
#include "GIParameterList.h"
#include "GIStringP.h"
#include "GILongP.h"
#define MAXLEN 1024

try
{
    EcUtStatus status;

    GIClient* client = GIClient::Create(status,(char *) cuserid(0));
    if (status.Ok() == EcDFalse)
    {
        throw "GIClient::Create error";
    }
    EcTChar buffer[MAXLEN];

    // Identify the UR the local Data Server
    //
    cout << "Enter Data Server UR: "; cin.getline(buffer,MAXLEN);

    DsShSciServerUR server;
    istrstream is(buffer);
    is >> server;
    DsCIESDTReferenceCollector* collector =
    DsCIESDTReferenceCollector::Create(status,server,*client);
    if (status.Ok() == EcDFalse)
    {
        throw "DsCIESDTReferenceCollector::Create error";
    }

    // What Data Type does this granule belong to?
    //
    cout << "Enter ESDT ShortName: "; cin.getline(buffer,MAXLEN);

    DsShTypeID datatype(buffer);
```

```

// Define the location/name of the science granule file
//
cout << "Enter datafile: "; cin.getline(buffer,MAXLEN);

RWCString datafile = buffer;
RWTPtrOrderedVector<RWCString> datafiles;
datafiles.insert(&datafile);

// Define the location/name of the associated metadata file
//
cout << "Enter metafile: "; cin.getline(buffer,MAXLEN);

RWCString metafile = buffer;
DsClInsertCommand* insert = new DsClInsertCommand(datatype);
insert->AddMainGroup(metafile,datafiles);
DsEShSciPriority priority = DsShSciGlobal::HIGH;
DsClRequest request(DsClCommand &) *insert,priority);
status = request.Submit(collector);
if (status.Ok() == EcDFalse)
{
    throw "DsClRequest::Submit error";
}
const GIParameterList& results = request.GetResults();
results.saveOn(cerr); cerr << endl;
GIParameterList* cmdresults = (GIParameterList *) results.at(0);
GILongP* cmdstatus =
(GLongP *) cmdresults->FindParameter(DsCShCmdStatusP);
if (cmdstatus && cmdstatus->value())
{
    GIParameterList* insresults =
(GLParameterList *) cmdresults->at(0);
    GIStringP* UR =
(GLStringP *) insresults->FindParameter(DsCGeUR);
    UR->saveOn(cerr); cerr << endl;
    RWCString granuleUR = UR->value();

    // Display the resultant granule UR
    //
    cout << "granuleUR=" << granuleUR << endl;
}

else
{
    throw "error inserting data";
}
datafiles.clearAndDestroy();
delete insert;
delete collector;
delete client;
}

catch (EcTChar* message)

```

```

{
    cerr << "exception: " << message << endl;
}
catch (...)
{
    cerr << "unexpected runtime error" << endl;
}

```

End of coding example for inserting science granule(s).

- Science and Browse Insert - This interface is a variant of the above which provides for the insertion of both a science granule, and an associated browse file, at the same time. This effectively creates a multifile granule with a browse component. Note that only a single UR is returned for the collection.
- Science and QA Insert - This interface is another variant of the science insert which provides for the insertion of both a science granule, and an associated quality assurance file, at the same time. This effectively creates a multifile granule with a QA component. Again note that only a single UR is returned for the collection.
- Science, Browse and QA Insert - This interface is yet another variant of the science insert which provides for the insertion of a science granule, a quality assurance file and an associated browse file, at the same time. This effectively creates a multifile granule with both a browse and a QA component. Once again note that only a single UR is returned for the collection.
- Production History Insert - This interface is a specialized form of the science insert which provides for the insertion of a single production history tar file. As part of the interface, the user must specify a list of URs which point to the original science granules that were generated during the particular production run. Note that all of the science granules have to be inserted first, using one of the above interfaces, in order to retrieve the URs which will make this call possible. The purpose here is to create a back-reference to the Production history file for each science granule. As you'll see below, this allows the correct production history file to be retrieved for any single science granule.

Coding example for inserting production history:

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <sstream.h>
#include "EcPfClient.h"
#include "DsShSciServerUR.h"
#include "DsCIESDTReferenceCollector.h"
#include "DsCIRequest.h"
#include "DsCIInsertCommand.h"
```

```

#include "GIClient.h"
#include "GIParameterList.h"
#include "GIStringP.h"
#include "GILongP.h"
#define MAXLEN 1024

try
{
    EcUtStatus status;

    GIClient* client = GIClient::Create(status,(char *) cuserid(0));
    if (status.Ok() == EcDFalse)
    {
        throw "GIClient::Create error";
    }
    EcTChar buffer[MAXLEN];

    // Identify the UR the local Data Server
    //
    cout << "Enter server UR: "; cin.getline(buffer,MAXLEN);

    DsShSciServerUR server;
    istrstream is(buffer);
    is >> server;
    DsCIESDTReferenceCollector* collector =
    DsCIESDTReferenceCollector::Create(status,server,*client);
    if (status.Ok() == EcDFalse)
    {
        throw "DsCIESDTReferenceCollector::Create error";
    }
    DsShTypeID datatype("PH");

    // Define the location/name of the Production History tar file
    //
    cout << "Enter Production History datafile: "; cin.getline(buffer,MAXLEN);

    RWCString pdatafile = buffer;
    RWTPtrOrderedVector<RWCString> pdatafiles;
    pdatafiles.insert(&pdatafile);
    RWTPtrOrderedVector<RWCString> URs;
    for (;;)
    {

        // Enter the UR for each associated science granule
        //
        cout << "Enter granule UR [or <CR>]: "; cin.getline(buffer,MAXLEN);

        if (strlen(buffer) != 0)
        {
            URs.insert(new RWCString(buffer));
        }
        else

```

```

    {
        break;
    }
}

DsClInsertCommand* insert = new DsClInsertCommand(datatype);
insert->AddMainGroup(pdatafiles);
if (URs.entries() != 0)
    insert->AddAssocGroup(URs,DsClInsertCommand::UR);
DsEShSciPriority priority = DsShSciGlobal::HIGH;
DsClRequest request((DsClCommand &) *insert,priority);
status = request.Submit(collector);
if (status.Ok() == EcDFalse)
{
    throw "DsClRequest::Submit error";
}
const GIParameterList& results = request.GetResults();
results.saveOn(cerr); cerr << endl;
GIParameterList* cmdresults = (GIParameterList *) results.at(0);
GILongP* cmdstatus =
(GLongP *) cmdresults->FindParameter(DsCShCmdStatusP);
if (cmdstatus && cmdstatus->value())
{
    GIParameterList* insresults =
(GLParameterList *) cmdresults->at(0);
GIStringP* UR =
(GLStringP *) insresults->FindParameter(DsCGeUR);
UR->saveOn(cerr); cerr << endl;
RWCString granuleUR = UR->value();

// Display the resultant Production History UR
//
cout << "granuleUR=" << granuleUR << endl;

}
else
{
    throw "error inserting data";
}
URs.clearAndDestroy();
pdatafiles.clearAndDestroy();
delete insert;
delete collector;
delete client;
}

catch (EcTChar* message)
{
    cerr << "exception: " << message << endl;
}
catch (...)
{
    cerr << "unexpected runtime error" << endl;
}

```

```
}
```

End of coding example for inserting production history.

8.5.2 Acquire Interfaces

Acquire interfaces are used by:

Data Manager

Execution Manager

QA Monitor

This set of interfaces provides the essential services for the “staging” of files from the IMF archive to the local science platform. Again, various types of files, or collections of files, can be acquired, but all such acquires are performed on an ESDT basis. That is, only files belonging to a specific data type can be acquired at any one time. Upon return to acknowledge the successful acquisition of data, a parameter list containing the names of the acquired granule(s), and their associated metadata files, is returned to the caller. These identifiers are essential in managing the files while they reside on the local science platform.

Note that the calling sequence shown in the production history acquire indicates that granules are “pushed” from the archive using “ftp”. However, this is a vestige of the old Release A Science Data Server design. In the IMF, all file transfers are performed via NFS copies.

- Science Granule Acquire - This is the standard interface used within PDPS to retrieve all science data granules from the IMF Archive. All such acquires are performed on an ESDT for a given collection time range. As a result, more than one science granule may be retrieved in the process.

Coding example for acquiring science granule(s):

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <strstream.h>
#include "EcPfClient.h"
#include "DsShSciServerUR.h"
#include "DsCIESDTReferenceCollector.h"
#include "DsCIRequest.h"
#include "DsCIInsertCommand.h"
#include "GIClient.h"
#include "GIParameterList.h"
#include "GIStringP.h"
#include "GILongP.h"
#define MAXLEN 1024

try
{
    EcUtStatus status;
```

```

GlClient* client = GlClient::Create(status,(char *) cuserid(0));
if (status.Ok() == EcDFalse)
{
    throw "GlClient::Create error";
}
EcTChar buffer[MAXLEN];

// Identify the UR the local Data Server
//
cout << "Enter granule UR: "; cin.getline(buffer,MAXLEN);

DsShESDTUR UR;
istrstream ig(buffer);
ig >> UR;
DsCIESDTReference* granule =
DsCIESDTReference::Create(status,UR,*client);
if (status.Ok() == EcDFalse)
{
    throw "DsCIESDTReference::Create error";
}
RWCString username = (char *) cuserid(0);
RWCString password;
sprintf(buffer,"Enter password for %s: ",username.data());
password = getpass(buffer);
RWCString host;

// Identify the destination host for the acquired file(s)
//
cout << "Enter host: "; cin.getline(buffer,MAXLEN);

host = buffer;
RWCString destination;

// Identify the destination directory for the acquired file(s)
//
cout << "Enter destination: "; cin.getline(buffer,MAXLEN);

destination = buffer;
DsClAcquireCommand* acquire = DsClAcquireCommand::Create(status);
if (status.Ok() == EcDFalse)
{
    throw "DsClAcquireCommand::Create error";
}
RWCString mediaformat = "FILEFORMAT";
RWCString userProfID;
acquire->SetForFtpPushNoNotify(mediaformat,userProfID,username,
                                password,host,destination);
DsEShSciPriority priority = DsShSciGlobal::HIGH;
DsClRequest request((DsClCommand &) *acquire,priority);
status = granule->Submit(request);
if (status.Ok() == EcDFalse)

```

```

{
    throw "DsCIESDTReference::Submit error";
}
const GIParameterList& results = request.GetResults();
results.saveOn(cerr); cerr << endl;
GIParameterList* cmdresults = (GIParameterList *) results.at(0);
GILongP* cmdstatus =
(GILongP *) cmdresults->FindParameter(DsCShCmdStatusP);
GIParameterList* acqresults =
(GIParameterList *) cmdresults->FindParameter("UnnamedPL");
GIParameter* failure = acqresults ?
(GIParameter *) acqresults->FindParameter("FAILURE") : 0;
if (cmdstatus && cmdstatus->value() && acqresults && !failure)
{
    RWCString ftphost =
((GILStringP *) acqresults->FindParameter("FTPHOST"))->value();

    // Display the resultant host as verification of the request
    //
    cout << "ftphost=" << ftphost << endl;

    RWCString ftpdir =
((GILStringP *) acqresults->FindParameter("FTPPDIR"))->value();

    // Display the resultant directory as verification of the request
    //
    cout << "ftpdir=" << ftpdir << endl;

    RWCString granule;
    RWCString esdt;
    RWCString filename;
    RWCString filesize;
    GIParameter* parameter;
    for (EcTInt i = 0;i < acqresults->entries();++i)
    {
        parameter = (GIParameter *) acqresults->at(i);
        if (parameter->GetName() == "GRANULE")
        {
            granule = ((GILStringP *) parameter)->value();

            // Display the Data Server granule identifier
            //
            cout << "granule=" << granule << endl;

        }
        else if (parameter->GetName() == "ESDT")
        {
            esdt = ((GILStringP *) parameter)->value();

            // Display the data type (ESDT) of the acquired granule
            //
            cout << "esdt=" << esdt << endl;
        }
    }
}

```

```

    }

    else if (parameter->GetName() == "FILENAME")
    {
        filename = ((GlStringP *) parameter)->value();

        // Display the name of the acquired granule
        //
        cout << "filename=" << filename << endl;

    }

    else if (parameter->GetName() == "FILESIZE")
    {
        filesize = ((GlStringP *) parameter)->value();

        // Display the size in Mbytes of the acquired granule
        //
        cout << "filesize=" << filesize << endl;

    }

}

else
{
    throw "error acquiring data";
}

delete acquire;
delete granule;
delete client;
}

catch (EcTChar* message)
{
    cerr << "exception: " << message << endl;
}
catch (...)
{
    cerr << "unexpected runtime error" << endl;
}

```

End of coding example for acquiring science granule(s).

- Browse Acquire - This variant of the standard interface, allows an associated browse file to be retrieved from the science granule UR, provided that a browse file was associated with the science granule during a combined insert call.
- QA Acquire - This variant of the standard interface, allows an associated quality assurance file to be retrieved from the science granule UR, provided that a QA file was associated with the science granule during a combined insert call.

- Production History Acquire - As mentioned above, this interface is used by the QA Monitor application to retrieve the production history tar file for a particular science granule. Though many science granules can be associated with a production history file, the UR for any one of them may be used to retrieve this file.

Coding example for acquiring production history:

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <strstream.h>
#include "EcPfClient.h"
#include "DsShSciServerUR.h"
#include "DsCIESDTReferenceCollector.h"
#include "DsCIRequest.h"
#include "DsCIInsertCommand.h"
#include "GIClient.h"
#include "GIParameterList.h"
#include "GIStringP.h"
#include "GILongP.h"
#define MAXLEN 1024

try
{
    EcUtStatus status;

    GIClient* client = GIClient::Create(status,(char *) cuserid(0));
    if (status.Ok() == EcDFalse)
    {
        throw "GIClient::Create error";
    }
    EcTChar buffer[MAXLEN];

    // Identify the UR the local Data Server
    //
    cout << "Enter granule UR: "; cin.getline(buffer,MAXLEN);

    DsShESDTUR UR;
    istrstream ig(buffer);
    ig >> UR;
    DsCIESDTReference* granule =
    DsCIESDTReference::Create(status,UR,*client);
    if (status.Ok() == EcDFalse)
    {
        throw "DsCIESDTReference::Create error";
    }
    RWTPtrOrderedVector<RWCString> attributes;
    attributes.insert(new RWCString(DsCGeProdHistoryUR));
    GIParameterList iresults;
    status = granule->Inspect(attributes,iresults);
    if (status.Ok() == EcDFalse)
    {
```

```

        throw "DsClESDTReference::Inspect error";
    }
    GIStringP* phID =
    (GIStringP *) iresults.FindParameter(DsCGeProdHistoryUR);
    if (!phID)
    {
        throw "no productionHistoryId found";
    }
    delete granule;
    DsShESDTUR pUR(UR.GetServerUR(),phID->value());
    granule = DsClESDTReference::Create(status,pUR,*client);
    RWCString username = (char *) cuserid(0);
    RWCString password;
    sprintf(buffer,"Enter password for %s: ",username.data());
    password = getpass(buffer);
    RWCString host;

    // Identify the destination host for the acquired file(s)
    //
    cout << "Enter host: "; cin.getline(buffer,MAXLEN);

    host = buffer;
    RWCString destination;

    // Identify the destination directory for the acquired file(s)
    //
    cout << "Enter destination: "; cin.getline(buffer,MAXLEN);

    destination = buffer;
    DsClAcquireCommand* acquire = DsClAcquireCommand::Create(status);
    if (status.Ok() == EcDFFalse)
    {
        throw "DsClAcquireCommand::Create error";
    }
    RWCString mediaformat = "FILEFORMAT";
    RWCString userProfID;
    acquire->SetForFtpPushNoNotify(mediaformat,userProfID,username,
                                    password,host,destination);
    DsEShSciPriority priority = DsShSciGlobal::HIGH;
    DsClRequest request((DsClCommand &) *acquire,priority);
    status = granule->Submit(request);
    if (status.Ok() == EcDFFalse)
    {
        throw "DsClESDTReference::Submit error";
    }
    const GIParameterList& results = request.GetResults();
    results.saveOn(cerr); cerr << endl;
    GIParameterList* cmdresults = (GIParameterList *) results.at(0);
    GILongP* cmdstatus =
    (GILongP *) cmdresults->FindParameter(DsCShCmdStatusP);
    GIParameterList* acqresults =
    (GIParameterList *) cmdresults->FindParameter("UnnamedPL");

```

```

GIParameter* failure = acqresults ?
(GIParameter *) acqresults->FindParameter("FAILURE") : 0;
if (cmdstatus && cmdstatus->value() && acqresults && !failure)
{
    RWCString ftphost =
((GIStringP *) acqresults->FindParameter("FTPHOST"))->value();

    // Display the resultant host as verification of the request
    //
    cout << "ftphost=" << ftphost << endl;

    RWCString ftpdir =
((GIStringP *) acqresults->FindParameter("FTPDIR"))->value();

    // Display the resultant directory as verification of the request
    //
    cout << "ftpdir=" << ftpdir << endl;

    RWCString granule;
    RWCString esdt;
    RWCString filename;
    RWCString filesize;
    GIParameter* parameter;
    for (EcTInt i = 0;i < acqresults->entries();++i)
    {
        parameter = (GIParameter *) acqresults->at(i);
        if (parameter->GetName() == "GRANULE")
        {
            granule = ((GIStringP *) parameter)->value();

            // Display the Data Server granule identifier
            //
            cout << "granule=" << granule << endl;

        }
        else if (parameter->GetName() == "ESDT")
        {
            esdt = ((GIStringP *) parameter)->value();

            // Display the data type (ESDT) of the acquired granule
            //
            cout << "esdt=" << esdt << endl;

        }
        else if (parameter->GetName() == "FILENAME")
        {
            filename = ((GIStringP *) parameter)->value();

            // Display the name of the acquired granule
            //
            cout << "filename=" << filename << endl;

```

```

        }
        else if (parameter->GetName() == "FILESIZE")
        {
            filesize = ((GlStringP *) parameter)->value();

            // Display the size in Mbytes of the acquired granule
            //
            cout << "filesize=" << filesize << endl;

        }
    }
else
{
    throw "error acquiring data";
}
irequests.RecursiveDelete();
attributes.clearAndDestroy();
delete acquire;
delete granule;
delete client;
}

catch (EcTChar* message)
{
    cerr << "exception: " << message << endl;
}
catch (...)
{
    cerr << "unexpected runtime error" << endl;
}

```

End of coding example for acquiring production history.

8.5.3 Inspect and Query Interfaces

Inspect and Query Interfaces are used by:

Production Request Editor

Data Manager

Execution Manager

QA Monitor

This set of interfaces provides services for inspecting data granules for key metadata attribute information, and for retrieving the URs of those granules which match selected metadata attribute query criteria. The inspection interfaces return parameter lists of attribute value information, while

the query interfaces return the URs for granules of interest.

- Science Granule Inspect - This interface is used to retrieve one or more metadata attribute values that are associated with a particular science granule. Only the values of those attributes are returned. Some of the attributes which are commonly inspected, within the PDPS, are the collection start and stop times of the granules

Coding example for inspecting science granule:

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <istrstream.h>
#include "EcPfClient.h"
#include "DsShSciServerUR.h"
#include "DsCIESDTReferenceCollector.h"
#include "DsCIRequest.h"
#include "DsCIInsertCommand.h"
#include "GIClient.h"
#include "GIParameterList.h"
#include "GIStringP.h"
#include "GILongP.h"

#define MAXLEN 1024

try
{
    EcUtStatus status;

    GIClient* client = GIClient::Create(status,(char *)cuserid(0));
    if (status.Ok() == EcDFalse)
    {
        throw "GIClient::Create error";
    }
    EcTChar buffer[MAXLEN];

    // Identify the UR the local Data Server
    //
    cout << "Enter granule UR: "; cin.getline(buffer,MAXLEN);

    DsShESDTUR UR;
    istrstream ig(buffer);
    ig >> UR;
    DsCIESDTReference* granule =
    DsCIESDTReference::Create(status,UR,*client);
    if (status.Ok() == EcDFalse)
    {
        throw "DsCIESDTReference::Create error";
    }

    // Test for queryable parameters on this granule
    //
```

```

const GIParameterList* queryablelist =
granule->GetQueryableParameters(DsShSciGlobal::GRANULE);
if (!queryablelist || queryablelist->entries() == 0 ||
    ((GIParameterList *) queryablelist->at(0))->entries() == 0)
    cerr << "no queryable parameters found" << endl;
else
{
    queryablelist->saveOn(cerr); cerr << endl;
}

RWTPtrOrderedVector<RWCString> attributes;
for (;;)
{
    // Identify one, or more, attributes to inspect
    //
    cout << "Enter attribute [or <CR>]: "; cin.getline(buffer,MAXLEN);

    if (strlen(buffer) != 0)
    {
        attributes.insert(new RWCString(buffer));
    }
    else
    {
        break;
    }
}
GIParameterList results;
status = granule->Inspect(attributes,results);
if (status.Ok() == EcDFalse)
{
    throw "DsCIESDTReference::Inspect error";
}
results.saveOn(cerr); cerr << endl;
GIParameter* parameter;
for (EcTInt i = 0;i < attributes.entries();++i)
{
    if (parameter =
        (GIParameter *) results.FindParameter(*attributes[i]))
    {
        parameter->saveOn(cerr); cerr << endl;
        if (parameter->isA() == GIClassType::DATEP)
        {

            // Display the value of a date type attribute
            //
            cout << "[DATEP]" << parameter->GetName() << "="
            << ((GIDateP *) parameter)->value() << endl;

        }
        else if (parameter->isA() == GIClassType::DOUBLEP)
        {

```

```

// Display the value of a double precision floating point attribute
//
cout << "[DOUBLEP]" << parameter->GetName() << "="
    << ((GlDoubleP *) parameter)->value() << endl;

}

else if (parameter->isA() == GlClassType::LONGP)
{

    // Display the value of a long integer attribute
    //
    cout << "[LONGP]" << parameter->GetName() << "="
        << ((GlLongP *) parameter)->value() << endl;

}

else if (parameter->isA() == GlClassType::STRINGP)
{

    // Display the value of a character string attribute
    //
    cout << "[STRINGP]" << parameter->GetName() << "="
        << ((GlStringP *) parameter)->value() << endl;

}

else if (parameter->isA() == GlClassType::TIMEP)
{

    // Display the value of a temporal attribute
    //
    cout << "[TIMEP]" << parameter->GetName() << "="
        << ((GlTimeP *) parameter)->value() << endl;

}

else
{

    // Note that the attribute is not associated with this granule
    //
    cout << *attributes[i] << " attribute not found" << endl;

}

results.RecursiveDelete();
attributes.clearAndDestroy();
delete granule;
delete client;
}

catch (EcTChar* message)
{

```

```

        cerr << "exception: " << message << endl;
    }
    catch (...)
    {
        cerr << "unexpected runtime error" << endl;
    }
}

```

End of coding example for inspecting science granule.

- ESDT Metadata Query - Similar to the inspect interface, this interface provides the means to retrieve one, or more science granules URs based on the results of a set of query conditions which are applied to the metadata attributes of a specific data type. The returned URs may then be used to acquire those science granules.

Coding example for querying inventory metadata:

```

#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <istrstream.h>
#include "EcPfClient.h"
#include "DsShSciServerUR.h"
#include "DsCIESDTReferenceCollector.h"
#include "DsCIRequest.h"
#include "DsCIInsertCommand.h"
#include "GlClient.h"
#include "GlParameterList.h"
#include "GlStringP.h"
#include "GlLongP.h"
#define MAXLEN 1024

try
{
    EcUtStatus status;

    GlClient* client = GlClient::Create(status,(char *) cuserid(0));
    if (status.Ok() == EcDFalse)
    {
        throw "GlClient::Create error";
    }
    EcTChar buffer[MAXLEN];

    // Identify the UR the local Data Server
    //
    cout << "Enter server UR: "; cin.getline(buffer,MAXLEN);

    DsShSciServerUR server;
    istrstream is(buffer);

```

```

is >> server;
DsCIESDTReferenceCollector* collector =
DsCIESDTReferenceCollector::Create(status,server,*client);
if (status.Ok() == EcDFalse)
{
    throw "DsCIESDTReferenceCollector::Create error";
}

// Identify the data type of the granules to be queried
//
cout << "Enter ShortName: "; cin.getline(buffer,MAXLEN);

GIStringP datatype(buffer,"ShortName");
datatype.SetDescription(":");

// Identify the data collection start time as the value of the first search criterion
//
cout << "Enter RangeBeginningDate {MM/DD/YY[YY]}: "; cin.getline(buffer,MAXLEN);

RWCString bdate = RWDate(buffer).asString();
GIStringP begindate(bdate,"RangeBeginningDate");
begindate.SetDescription(">=");

// Identify the data collection stop time as the value of the second search criterion
//
cout << "Enter RangeEndingDate {MM/DD/YY[YY]}: "; cin.getline(buffer,MAXLEN);

RWCString edate = RWDate(buffer).asString();
GIStringP enddate(edate,"RangeEndingDate");
enddate.SetDescription("<=");

GIParameterList constraints("Constraints");
constraints.insert(&datatype);
constraints.insert(&begindate);
constraints.insert(&enddate);
DsClQuery* query = DsClQuery::Create(status,&constraints);
if (status.Ok() == EcDFalse)
{
    throw "DsClQuery::Create error";
}
status = collector->Search(*query);
if (status.Ok() == EcDFalse)
{
    throw "DsCIESDTReferenceCollector::Search error";
}
if (collector->entries() == 0)
{
    throw "query yielded nothing!";
}
DsCIESDTReference* granule;
for (EcTInt i = 0;i < collector->entries();++i)
{

```

```

granule = (*collector)[i];

// Display the UR of the granule(s) which match the search criteria
//
cout << "[" << i << "]" << granule->Textify() << endl;
cout << "UR = " << granule->GetUR() << endl;

}

delete query;
delete collector;
delete client;
}

catch (EcTChar* message)
{
    cerr << "exception: " << message << endl;
}
catch (...)
{
    cerr << "unexpected runtime error" << endl;
}

```

End of coding example for querying inventory metadata.

- ESDT Insert Time Query - This interface, which is very similar to the previous one, provides the means to query on the IMF archive insertion time of science granules, which is not an inventory metadata attribute.

Coding example for querying on insert time:

```

#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <strstream.h>
#include "EcPfClient.h"
#include "DsShSciServerUR.h"
#include "DsCIESDTReferenceCollector.h"
#include "DsCIRequest.h"
#include "DsCIInsertCommand.h"
#include "GIClient.h"
#include "GIParameterList.h"
#include "GIStringP.h"
#include "GILongP.h"
#define MAXLEN 1024

try
{
    EcUtStatus status;

```

```

GlClient* client = GlClient::Create(status,(char *) cuserid(0));
if (status.Ok() == EcDFalse)
{
    throw "GlClient::Create error";
}
EcTChar buffer[MAXLEN];

// Identify the UR the local Data Server
//
cout << "Enter server UR: "; cin.getline(buffer,MAXLEN);

DsShSciServerUR server;
istrstream is(buffer);
is >> server;
DsCIESDTReferenceCollector* collector =
DsCIESDTReferenceCollector::Create(status,server,*client);
if (status.Ok() == EcDFalse)
{
    throw "DsCIESDTReferenceCollector::Create error";
}

// Identify the data type of the granules to be queried
//
cout << "Enter ShortName: "; cin.getline(buffer,MAXLEN);

GlStringP datatype(buffer,"ShortName");
datatype.SetDescription("=");
RWCString stamp;

// Identify the archive insertion starting range as the value of the first search criterion
//
cout << "Enter insertTime (begin) {MM/DD/YY[YY] [HH:MM:SS]}: "; cin.getline(buffer,MAXLEN);

stamp = buffer;
GlStringP begindate(stamp,"insertTime");
begindate.SetDescription(">=");

// Identify the archive insertion ending range as the value of the second search criterion
//
cout << "Enter insertTime (end) {MM/DD/YY[YY] [HH:MM:SS]}: "; cin.getline(buffer,MAXLEN);

stamp = buffer;
GlStringP enddate(stamp,"insertTime");
enddate.SetDescription("<=");

GlParameterList constraints("Constraints");
constraints.insert(&datatype);
constraints.insert(&begindate);
constraints.insert(&enddate);
DsCIQuery* query = DsCIQuery::Create(status,&constraints);
if (status.Ok() == EcDFalse)

```

```

{
    throw "DsCIQuery::Create error";
}
status = collector->Search(*query);
if (status.Ok() == EcDFalse)
{
    throw "DsCIESDTReferenceCollector::Search error";
}
if (collector->entries() == 0)
{
    throw "query yielded nothing!";
}
DsCIESDTReference* granule;
for (EcTInt i = 0;i < collector->entries();++i)
{
    granule = (*collector)[i];

    // Display the UR of the granule(s) which match the search criteria
    //
    cout << "[" << i << "]" << granule->Textify() << endl;
    cout << "UR = " << granule->GetUR() << endl;

}
delete query;
delete collector;
delete client;
}

catch (EcTChar* message)
{
    cerr << "exception: " << message << endl;
}
catch (...)
{
    cerr << "unexpected runtime error" << endl;
}

```

End of coding example for querying on insert time.

8.5.4 Update Interface

Metadata Update Interface is used by:

QA Monitor

This interface provides the service to perform an update of the inventory metadata that is associated with a specific science granule. In the Pre-Release B Testbed, this equates to modifying the metadata file within the IMF archive, for the granule of interest. Note that only the QA Monitor application has the requirement, and ability to perform such updates; even then, those updates are

restricted to the set of QA collection flag attributes, commonly found within the inventory metadata of most science granules.

Coding example for updating metadata:

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <strstream.h>
#include "EcPfClient.h"
#include "DsShSciServerUR.h"
#include "DsCIESDTReferenceCollector.h"
#include "DsCIRequest.h"
#include "DsCIInsertCommand.h"
#include "GIClient.h"
#include "GIParameterList.h"
#include "GIStringP.h"
#include "GILongP.h"
#define MAXLEN 1024

try
{
    EcUtStatus status;

    GIClient* client = GIClient::Create(status,(char *) cuserid(0));
    if (status.Ok() == EcDFalse)
    {
        throw "GIClient::Create error";
    }
    EcTChar buffer[MAXLEN];

    // Identify the UR the local Data Server
    //
    cout << "Enter granule UR: "; cin.getline(buffer,MAXLEN);

    DsShESDTUR UR;
    istrstream ig(buffer);
    ig >> UR;
    DsCIESDTReference* granule =
    DsCIESDTReference::Create(status,UR,*client);
    if (status.Ok() == EcDFalse)
    {
        throw "DsCIESDTReference::Create error";
    }
    GIParameterList attributes;
    RWCString name;
    RWCString value;
    for (;;)
    {

        // Identify the attribute name of the metadata to be updated
        //
```

```

cout << "Enter attribute name: "; cin.getline(buffer,MAXLEN);

if (strlen(buffer) != 0)
{
    name = buffer;

    // Identify the attribute value of the metadata to be updated
    //
    cout << "Enter attribute value: "; cin.getline(buffer,MAXLEN);

    value = buffer;
    GIStringP* attribute = new GIStringP(value,name);
    attributes.insert(attribute);
}

else
{
    break;
}
}

DsClCommand* update =
new DsClCommand(DsCGeMetadataUpdate,attributes,DsShSciGlobal::ESDT);
DsEShSciPriority priority = DsShSciGlobal::HIGH;
DsClRequest request(*update,priority);
status = granule->Submit(request);
if (status.Ok() == EcDFalse)
{
    throw "DsClESDTReference::Submit error";
}
const GIParameterList& results = request.GetResults();
results.saveOn(cerr); cerr << endl;
GIParameterList* cmdresults = (GIParameterList *) results.at(0);
GILongP* cmdstatus =
(GLongP *) cmdresults->FindParameter(DsCShCmdStatusP);
GLongP* esdtstatus =
(GLongP *) cmdresults->FindParameter(DsCSrESDTStatusP);
if (!cmdstatus || !cmdstatus->value() ||
!esdtstatus || !esdtstatus->value())
{
    throw "error updating metadata";
}

// The metadata update was performed successfully if this point is reached
//


attributes.clearAndDestroy();
delete update;
delete granule;
delete client;
}

catch (EcTChar* message)
{

```

```

        cerr << "exception: " << message << endl;
    }
    catch (...)
    {
        cerr << "unexpected runtime error" << endl;
    }
}

```

End of coding example for updating metadata.

8.5.5 Subscription Interface (Subscription Manager)

The Subscription Interface is used by:

SSIT Manger

This interface is used solely by the SSIT Manager to register subscriptions for science granules which belong to specific ESDTs. The purpose of which is to ensure that the arrival of such data (i.e., within the IMF archive) is announced to the interested parties (e.g., PDPS) via a notification event. These events are responsible for triggering the execution of PGEs that are waiting for the arrival of external data.

8.6 IMF Components

Primarily, the IMF software consists of the original Release A Science Data Server software (SDSRV), except for the DsGl and DsCl CSCs, where many of the classes have had member methods overridden to provide the essential hooks to the UNIX file system. Also, an additional class “DsCLIMF” has been introduced to provide for metadata access and validation in lieu of that functionality which would have been performed by the Data Server inventory services in the Release A system (e.g., metadata validation on insert). Since the Release A Science Data Server software is considered to be an OTS product (heritage code) for the Testbed, its design is not modeled and documented in this as-built documentation. However, high level operations and product architecture are documented in the Release A CDR design document.

The following is a list of Science Data Server classes and their methods which have been overridden by the IMF:

1. DsCIESDTReference Class

Objects of this class provide a reference to an ESDT that is within a Data Server's holdings.

The object provides services that are homogeneous for all ESDTs.

MakeMe()

This method is used by the client to create an instance from a UR.

2. DsCIESDTReferenceCollector Class

This class provides the primary interaction mechanism for client software. This class inherits DsClGenSuspendor to get the connection and basic collection behavior. This class contains the specialized functions that pertain to management of state (the working collection on the server side) by mimicking that state on the client machine.

DsClReferenceCollector()

This alternate constructor creates the DsClESDTReferenceCollector object, initializing the (inherited) attributes from the provided input.

3. DsClGenRequestInt Class

This class represents the generic portion of the client side's interface to request objects.

GenSubmit()

Submits to the DsClGenConnector.

4. DsClIMF Class

This class consists of PDPS methods for use as stubs to DSS.

OdlInit()

Open ODL file, read into memory.

UpdateFile()

Update ASCII metadata file with data in memory.

CheckMandatoryAttrs()

Check that MANDATORY attributes in a descriptor are present in the metadata file.

CheckProdSpecificAttrs()

Check that attributes in the descriptor group ProductSpecificMetadata are present in the metadata file.

CheckMetafileAttrs()

Check that all attributes in a metadata file are present in the descriptor file.

CheckBeginEndAttrs()

Check that a begin date/time is earlier than an end date/time in the meta file.

CheckExceptionalAttr()

Check whether an exceptional attribute is OK.

DateIsValid()

Check whether a string is a valid date.

TimeIsValid()

Check whether a string is a valid time.

AttrIsDate()

Check whether a string is a valid date attribute name.

AttrIsTime()

Check whether a string is a valid time attribute name.

FindOneAttribute()

Get metadata values for a single attribute from an MCF.

GetGroup()

Get metadata values for an entire group from an MCF.

UpdateGroupAttrs()

Update metadata values for a group.

UpdateOneAttribute()

Update metadata values for a single attribute.

DsCLIMF()

Default constructor.

DsCLIMF()

Alternate constructor; opens ODL files.

~DsCLIMF()

Destructor.

GetTmplAgg()

Get the value of attribute TmplAgg.

GetDescAgg()

Get the value of attribute DescAgg.

GetMetaAgg()

Get the value of attribute MetaAgg.

SetTmplAgg()

Set the value of attribute TmplAgg.

SetDescAgg()

Set the value of attribute DescAgg.

SetMetaAgg()

Set the value of attribute MetaAgg.

CheckMetadata()

Check metadata file content against ESDT descriptor file.

GetQueryableParameters()

Get metadata parameters from an ESDT descriptor file.

FindAttribute()

Get metadata values for a single attribute or group from a meta file.

UpdateAttribute()

Update metadata values for a parameter list or single parm in an MCF.

GetGroupQueryableParameters()

Get queryable metadata attributes from a single group, group GRANULEMETADATA or group COLLECTIONMETADATA, in the descriptor file.

5. DsClInsertCommand Class

This class provides an encapsulated mechanism for transporting ESDT insert command parameters through the SDSRV client interface.

AssAssocGroup()

Add a set of URs (required for valid Insert of Production History File).

6. DsClRequest Class

This is the object that clients submit to a data server to do anything. It is composed of a sequence of DsClCommand objects (created by the client), which are the basic execution units.

Submit()

Submit to the DsClESDTReferenceCollector, with an optional domain.

7. DsGlParameterList Class

This class provides a way to collect and manipulate GlParameter objects.

FindParameter()

This operation finds and returns a parameter by name.

DeepFindParameter()

This operation finds and returns a parameter by name no matter where it is in the parameter list. That is, it does a recursive search.

8. DsGlTimeP Class

This class provides a specialization of the GlParameter class for storing date data.

saveOn()

This operation outputs the information from this object to a normal stream in a human-readable format.

8.7 IMF Services, Configuration File, and Key Parameters

The few PDPS applications which require Data Server functionality are critical to the successful processing of science software. To ensure that these applications address the IMF properly, a configuration parameter has been established (for each application) which points the application to the correct version of the IMF by referencing a specific Data Server UR, (see Table 8.7-1). The following is a list of those PDPS applications which use IMF services and the corresponding configuration file which defines the key IMF parameter:

Table 8.7-1. Data Server UR

PDPS Application Program Name Configuration File	Parameter Name	Value
Production Request Editor PlPREditor PlPREditor_GSFC.CFG	DDSRV_UR	{Data Server UR}
Subscription Manager PlSubsReaper PlSubsReaper_GSFC.CFG	PlSubsMgr_DataServerHost	{IMF archive path}
Execution Manager DpPrEM DpPrEM_GSFC.CFG	DpPrDM_DataServerHost	{Data Server UR}
Data Manager DpPrDM DpPrDM_GSFC.CFG	DpPrDM_DataServerHost	{Data Server UR}
QA Monitor DpPrQaMonitorGUI DpPrQaMonitorGUI_GSFC.CFG	DpPrQA_DSS_HOST	{Data Server UR}
SSIT Manager : SSIT Executable Insert DpAtInsertExeTarFile DpAtIE_EDC.CFG	DataServerUr	{Data Server UR}
SSIT Manager : SSIT Static Insert DpAtInsertStaticFile DpAtIS_EDC.CFG	DataServerUr	{Data Server UR}
SSIT Manager : SSIT Dynamic Insert DpAtInsertTestFile DpAtID_EDC.CFG	DataServerUr	{Data Server UR}

This page intentionally left blank.

9. Testbed Databases

9.1 Overview

The Testbed operational database consists of planning and data processing database (pdps_db_ops), AutoSys COTS database (autosys), and the Sybase system database (master). Figure 9.1-1 illustrates the host machine at the DAAC site (i.e., one each at GSFC, LaRC, EDC, and NSIDC site), the SQL database servers, and the database names that have been created on those servers.

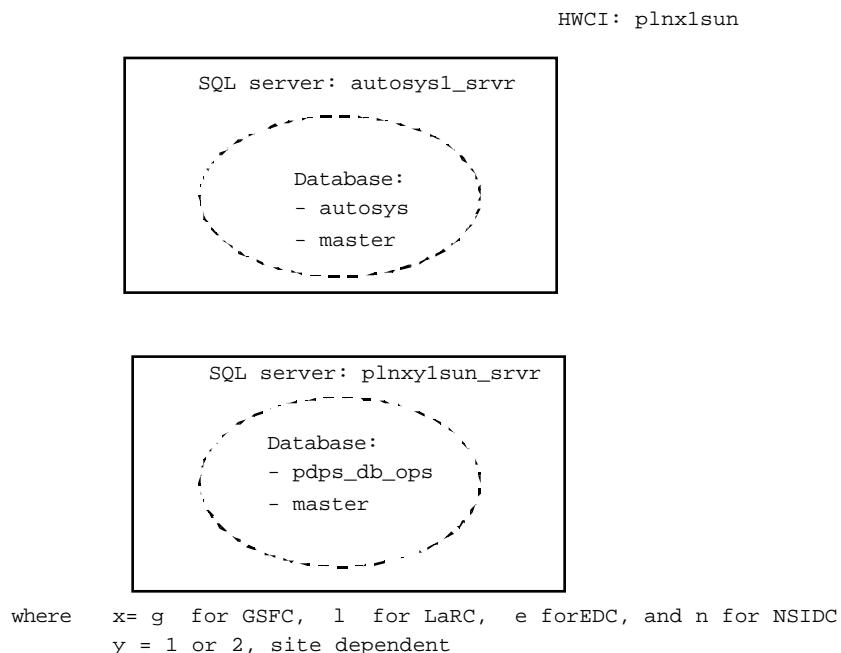


Figure 9.1-1 Testbed Database

The pdps_db_ops database is used for several functions. The Planning Subsystem uses the pdps_db_ops database to plan processing at the DAAC and resources which contains the information on configured resources and their allocation. The autosys database is used by AutoSys software to run science software and for controlling processing flows ('job') from one executable to another within the PGE and produce required products. Autosys application software is a COTS package, and it is one of the CSC of the Processing PRONG CSCI (an integral part of Data Processing Subsystem). The autosys database contains the information about the jobs and events, and it is used by AutoSys application software to provide job management function required for scheduling, monitoring, and reporting on science data production. The master database contains

Sybase system tables which are used to manage the existence and operation of SQL servers.

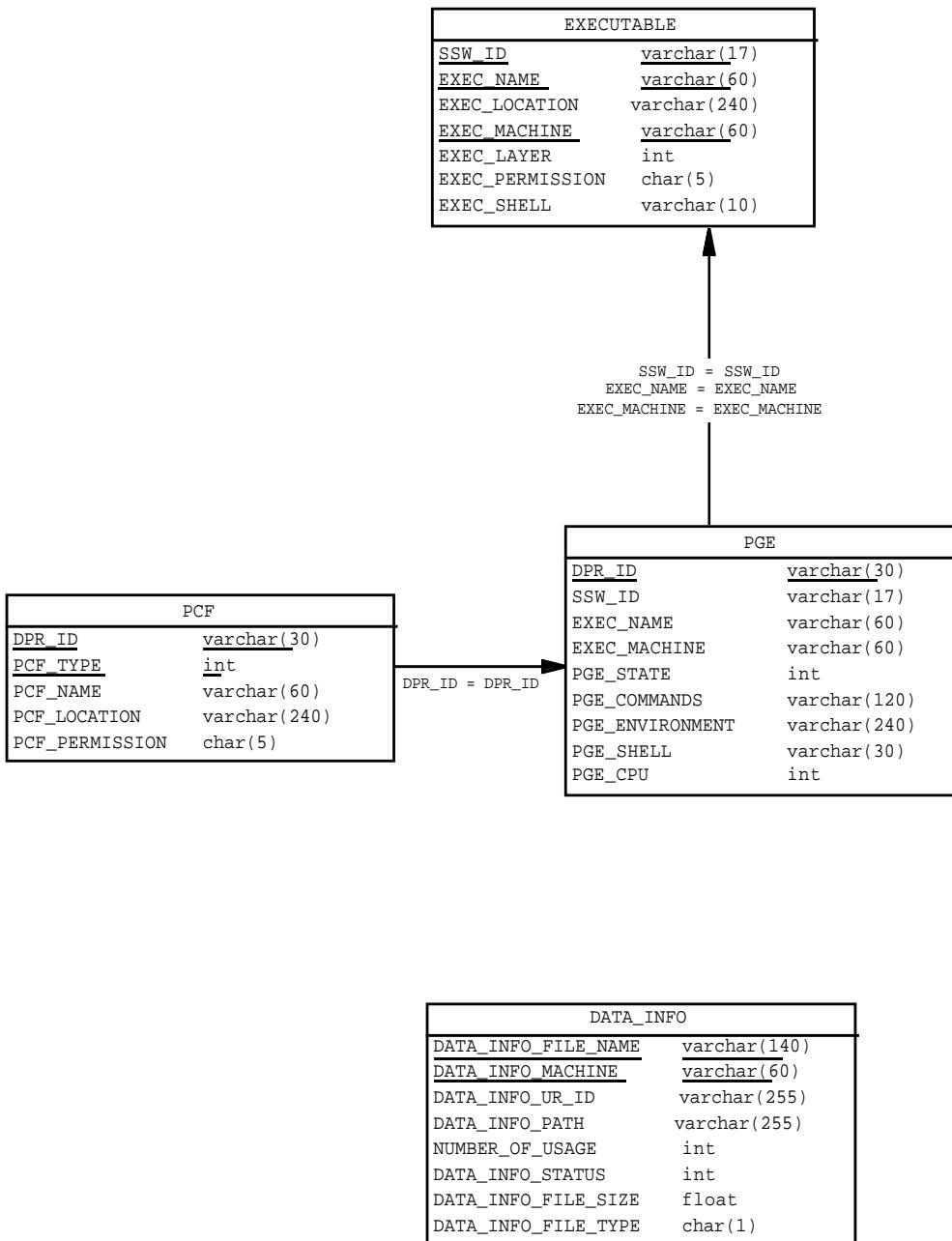
The SSI&T uses the capability provided by the testbed to support the integration and test of the science software (PGEs) within the production processing environment. As a part of this activity, the SSI&T GUIs provide for the entry into the databases for PGE-related information needed to plan for and run science software.

The following subsections provides a summary description of the pdps_db_ops database tables, and the tools used with the database. The autosys database schema has been described in the AutoSys COTS document.

9.2 Database Organization

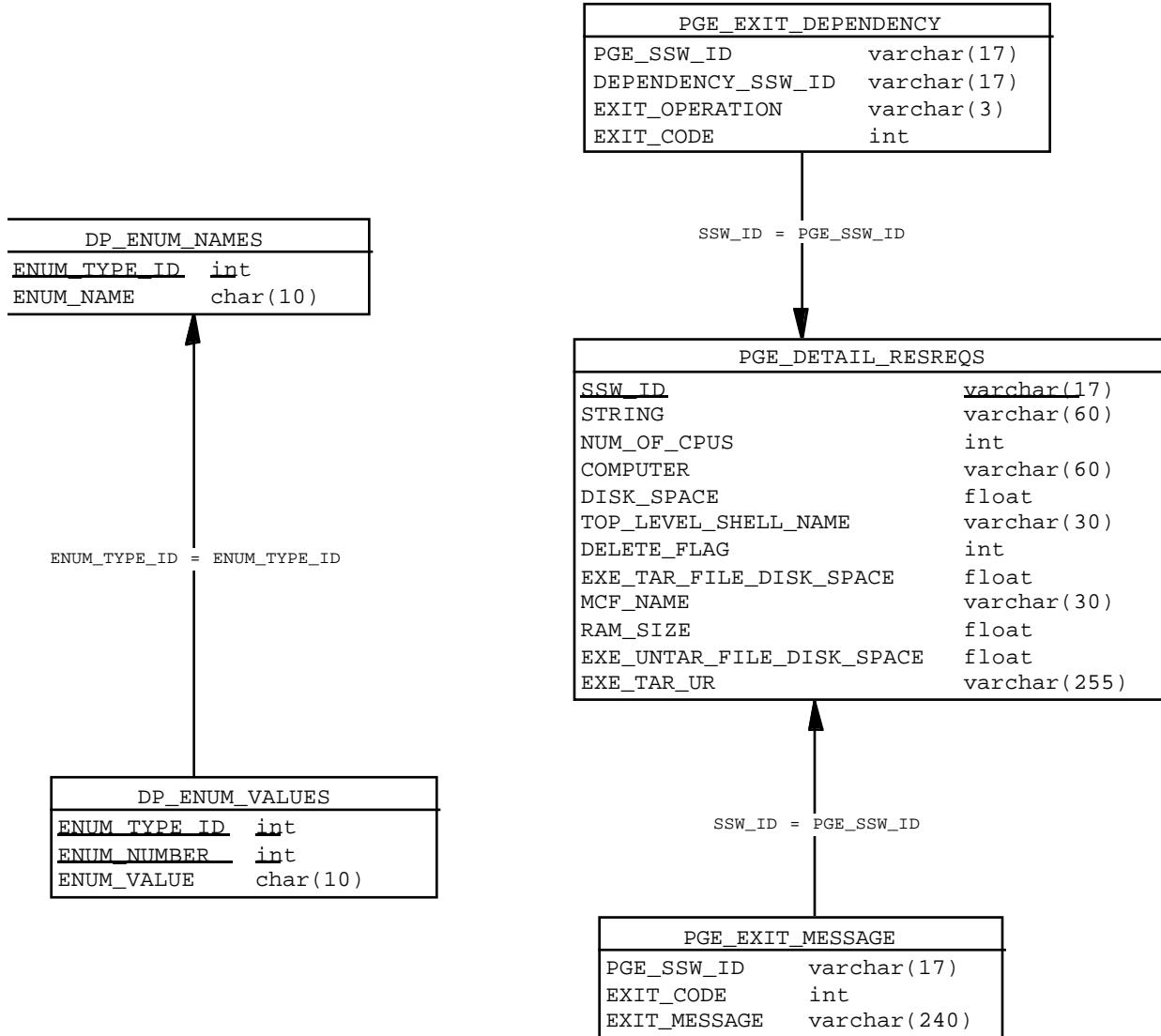
9.2.1 Schema

The database schema (physical data models) of pdps_db_ops database are provided in Figures 9.2.1-1 to 9.2.1-5 below. The underlined data shown in the figures represents the primary key.



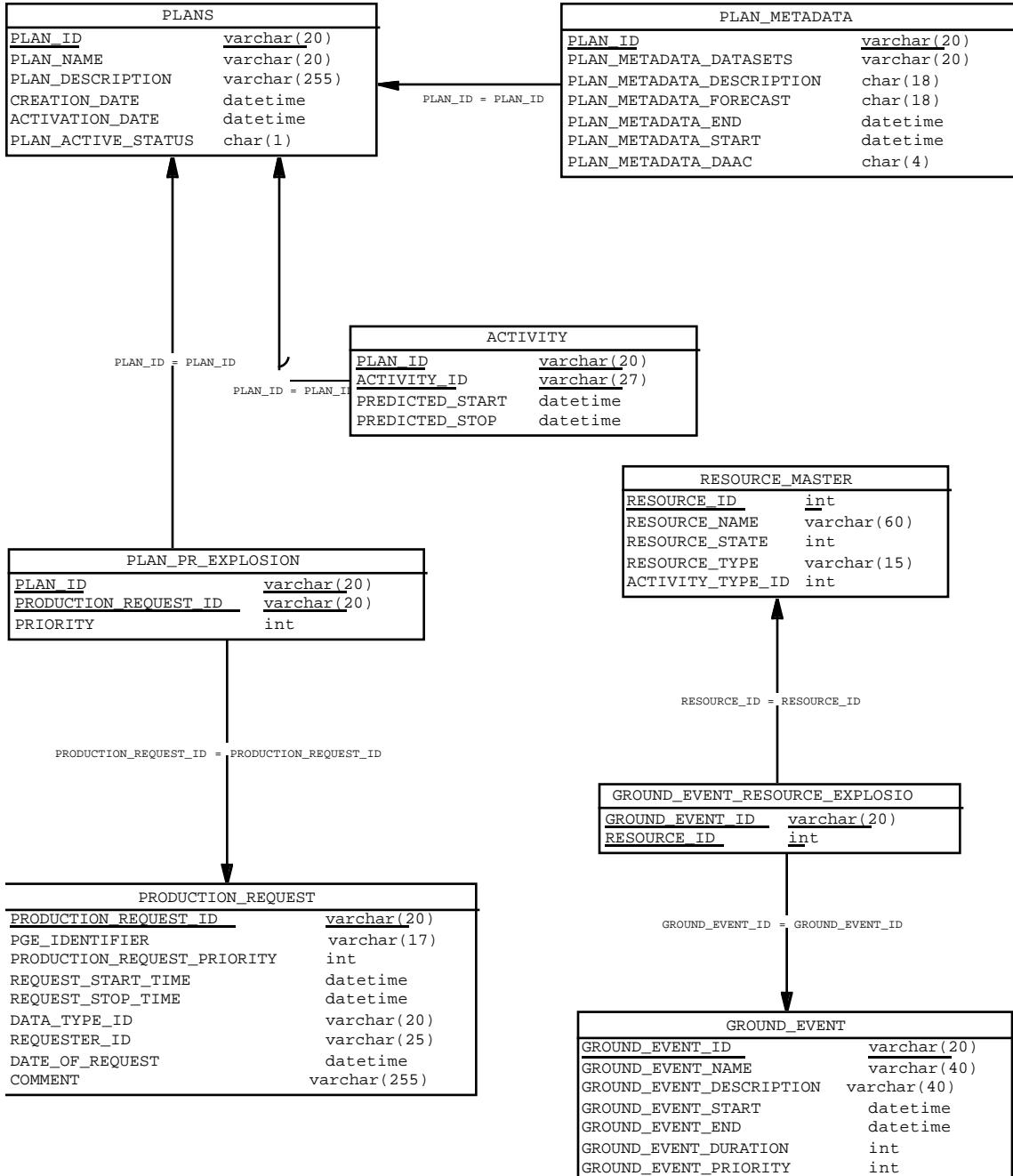
2

Figure 9.2.1-1 Execution and Data Management 1



2

Figure 9.2.1-2. Execution and Data Management 2



2

Figure 9.2.1-3. Production Plan

Figure 9.2.1-4 Production Request Edit

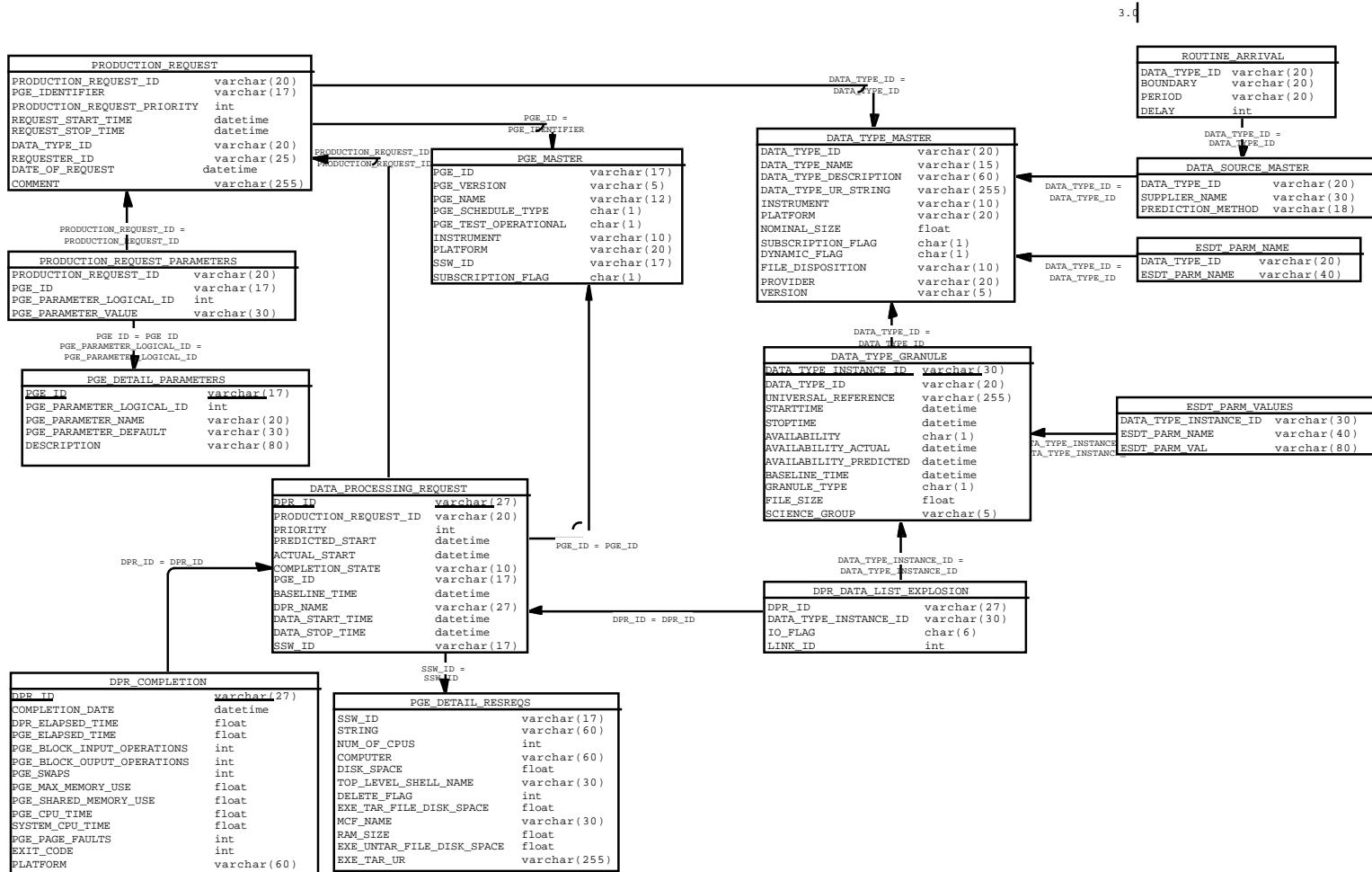
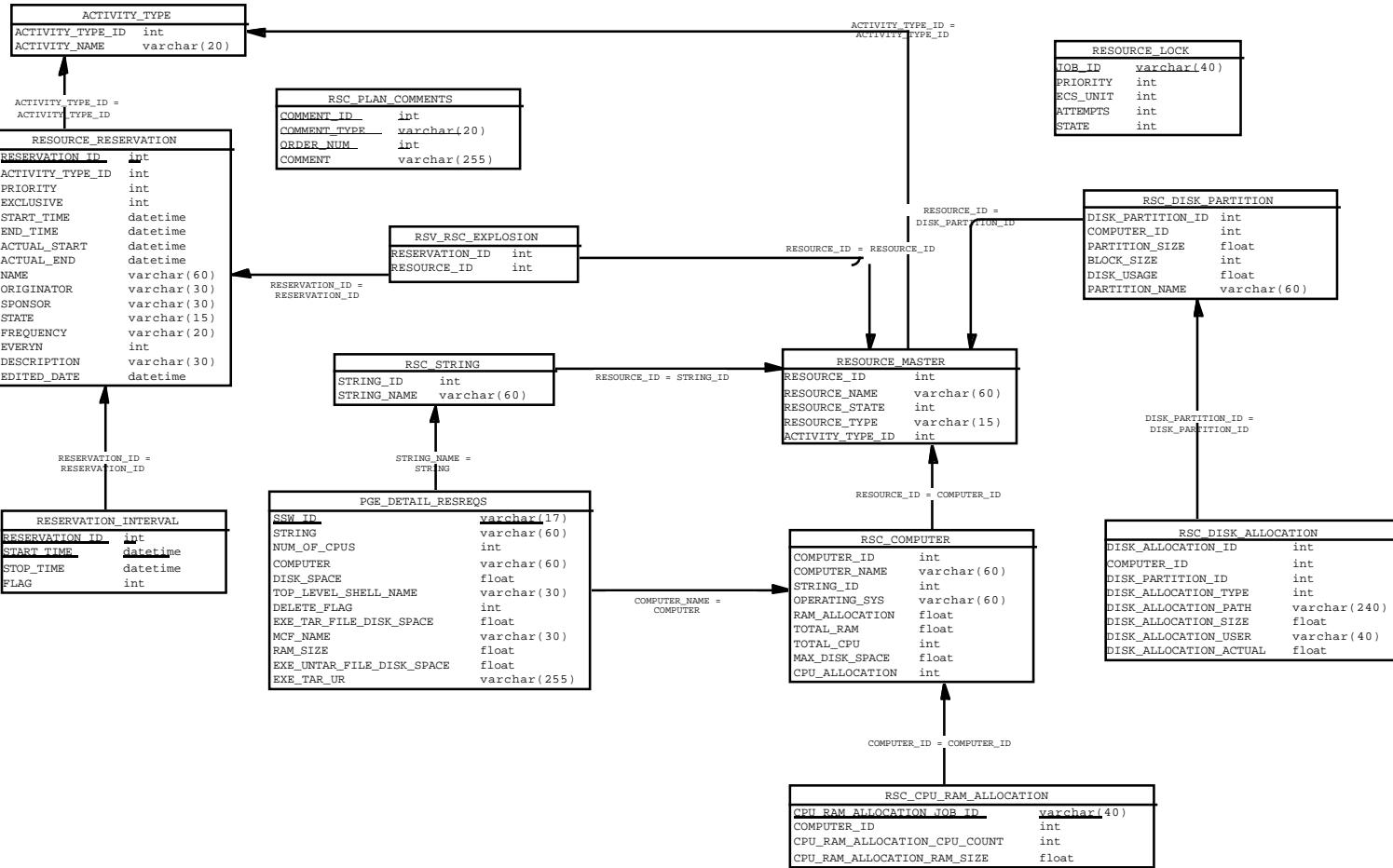


Figure 9.2.1-5 Resource Management

9.2.2 Tables and Columns

The following information shows the details of Sybase tables in the pdps_db_ops database:

Table 9.2.2-1 Database Tables and Columns

Database Table	Column	Description	Null/ Not Null
RESOURCE_LOCK	JOB_ID	varchar(40)	not null
	PRIORITY	int	not null
	ECS_UNIT	int	null
	ATTEMPTS	int	null
	STATE	int	null
RSC_PLAN_COMMENTS	COMMENT_ID	int	not null
	COMMENT_TYPE	varchar(20)	not null
	ORDER_NUM	int	null
	COMMENT	varchar(255)	null
RESOURCE_PLAN	RCS_PLAN_ID	int	not null
	RCS_PLAN_START_TIME	datetime	null
	RCS_PLAN_END_TIME	datetime	null
	RCS_PLAN_DESC	varchar(255)	null
DATA_INFO	DATA_INFO_FILE_NAME	varchar(140)	not null
	DATA_INFO_MACHINE	varchar(60)	not null
	DATA_INFO_UR_ID	varchar(255)	null
	DATA_INFO_PATH	varchar(255)	null
	NUMBER_OF_USAGE	int	null
	DATA_INFO_STATUS	int default 0 between 0 & 4	null
	DATA_INFO_FILE_SIZE	float	null
	DATA_INFO_FILE_TYPE	char(1)	null
PLANS	PLAN_ID	varchar(20)	not null
	PLAN_NAME	varchar(20)	null
	PLAN_DESCRIPTION	varchar(255)	null
	CREATION_DATE	datetime	null
	ACTIVATION_DATE	datetime	null
	PLAN_ACTIVE_STATUS	char(1)	null
DATA_TYPE_MASTER	DATA_TYPE_ID	varchar(20)	not null
	DATA_TYPE_NAME	varchar(15)	null
	DATA_TYPE_DESCRIPTION	varchar(60)	null
	DATA_TYPE_UR_STRING	varchar(255)	null
	INSTRUMENT	varchar(10)	null
	PLATFORM	varchar(20)	null
	NOMINAL_SIZE	float	null
	SUBSCRIPTION_FLAG	char(1)	null
	DYNAMIC_FLAG	char(1)	null
	FILE_DISPOSITION	varchar(10)	null
	PROVIDER	varchar(20)	null
	VERSION	varchar(5)	null

RSC_COMPUTER	COMPUTER_ID	int	not null
	COMPUTER_NAME	varchar(60)	not null
	STRING_ID	int	null
	OPERATING_SYS	varchar(60)	null
	RAM_ALLOCATION	float	null
	TOTAL_RAM	float	null
	TOTAL_CPU	int	null
	MAX_DISK_SPACE	float	null
	CPU_ALLOCATION	int	null
RSC_DISK_PARTITION	DISK_PARTITION_ID	int	not null
	COMPUTER_ID	int	null
	PARTITION_SIZE	float	null
	BLOCK_SIZE	int	null
	DISK_USAGE	float	null
	PARTITION_NAME	varchar(60)	null
PGE_DETAIL_RESREQS	SSW_ID	varchar(17)	not null
	STRING	varchar(60)	null
	NUM_OF_CPUS	int	null
	COMPUTER	varchar(60)	null
	DISK_SPACE	float	null
	TOP_LEVEL_SHELL_NAME	varchar(30)	null
	DELETE_FLAG	int	null
	EXE_TAR_FILE_DISK_SPACE	float	null
	MCF_NAME	varchar(30)	null
	RAM_SIZE	float	null
	EXE_UNTAR_FILE_DISK_SPACE	float	null
	EXE_TAR_UR	varchar(255)	null
PGE_MASTER	PGE_ID	varchar(17)	not null
	PGE_VERSION	varchar(5)	null
	PGE_NAME	varchar(12)	null
	PGE_SCHEDULE_TYPE	char(1)	null
	PGE_TEST_OPERATIONAL	char(1)	null
	INSTRUMENT	varchar(10)	null
	PLATFORM	varchar(20)	null
	SSW_ID	varchar(17)	not null
	SUBSCRIPTION_FLAG	char(1)	null
PGE_DETAIL_PARAMETERS	PGE_ID	varchar(17)	not null
	PGE_PARAMETER_LOGICAL_ID	int	not null
	PGE_PARAMETER_NAME	varchar(20)	null
	PGE_PARAMETER_DEFAULT	varchar(30)	null
	DESCRIPTION	varchar(80)	null
DATA_PROCESSING_REQUEST	DPR_ID	varchar(27)	not null
	PRODUCTION_REQUEST_ID	varchar(20)	not null
	PRIORITY	int	null
	PREDICTED_START	datetime	null
	ACTUAL_START	datetime	null
	COMPLETION_STATE	varchar(10)	null
	PGE_ID	varchar(17)	not null
	BASELINE_TIME	datetime	null
	DPR_NAME	varchar(27)	not null
	DATA_START_TIME	datetime	null
	DATA_STOP_TIME	datetime	null
	SSW_ID	varchar(17)	null

PRODUCTION_REQUEST	PRODUCTION_REQUEST_ID	varchar(20)	not null
	PGE_IDENTIFIER	varchar(17)	null
	PRODUCTION_REQUEST_PRIORITY	int	null
	REQUEST_START_TIME	datetime	null
	REQUEST_STOP_TIME	datetime	null
	DATA_TYPE_ID	varchar(20)	null
	REQUESTER_ID	varchar(25)	null
	DATE_OF_REQUEST	datetime	null
	COMMENT	varchar(255)	null
	PRODUCTION_REQUEST_STATE	varchar(10)	null
PGE	DPR_ID	varchar(30)	not null
	SSW_ID	varchar(17)	not null
	EXEC_NAME	varchar(60)	not null
	EXEC_MACHINE	varchar(60)	not null
	PGE_STATE	int	null
	PGE_COMMANDS	varchar(120)	null
	PGE_ENVIRONMENT	varchar(240)	null
	PGE_SHELL	varchar(30)	null
PLAN_METADATA	PGE_CPU	int	null
	PLAN_ID	varchar(20)	not null
	PLAN_METADATA_DATASETS	varchar(20)	null
	PLAN_METADATA_DESCRIPTION	char(18)	null
	PLAN_METADATA_FORECAST	char(18)	null
	PLAN_METADATA_END	datetime	null
	PLAN_METADATA_START	datetime	null
ACTIVITY	PLAN_METADATA_DAAC	char(4)	null
	PLAN_ID	varchar(20)	not null
	ACTIVITY_ID	varchar(27)	not null
	PREDICTED_START	datetime	null
	PREDICTED_STOP	datetime	null
DPR_DATA_LIST_EXPLOSION	DPR_ID	varchar(27)	not null
	DATA_TYPE_INSTANCE_ID	varchar(30)	not null
	IO_FLAG	char(6)	not null
	LINK_ID	int	null
PGE_PERFORMANCE	PGE_ID	varchar(17)	not null
	CPU_TIME	float	null
	PGE_ELAPSED_TIME	float	null
	DPR_ELAPSED_TIME	float	null
	MAX_MEMORY	float	null
	FAULTS	int	null
	SWAPS	int	null
	BLOCK_INPUT_OPERATION	int	null
	BLOCK_OUTPUT_OPERATION	int	null
	RUN_CPU_TIME	float	null
	RUN_PGE_ELAPSED	float	null
	RUN_MAX_MEMORY	float	null
	RUN_PAGE_FAULTS	int	null
	RUN_DPR_ELAPSED	float	null
	RUN_SWAPS	int	null
	RUN_BLOCK_IN_OPERATION	int	null
	RUN_BLOCK_OUT_OPERATION	int	null

ROUTINE_ARRIVAL	DATA_TYPE_ID	varchar(20)	not null
	BOUNDARY	varchar(20)	null
	PERIOD	varchar(20)	null
	DELAY	int	null
GROUND_EVENT_RESOURCE_EXPLOSION	GROUND_EVENT_ID	varchar(20)	not null
	RESOURCE_ID	int	not null
INPUT_DATA_SPECS	PGE_ID	varchar(17)	not null
	DATA_TYPE_ID	varchar(20)	not null
	LOGICAL_ID	int	not null
	DATA_TYPE_REQUIREMENT	smallint	null
	LINK_ID	int	null
	WHERE_CLAUSE	varchar(255)	null
	PCF_FILE_TYPE	int	null
	SCIENCE_GROUP	varchar(5)	null
OUTPUT_DATA_YIELD	PGE_ID	varchar(17)	not null
	DATA_TYPE_ID	varchar(20)	not null
	LOGICAL_ID	int	null
	YIELD	smallint	null
	LINK_ID	int	null
	SCIENCE_GROUP	varchar(5)	null
PGE_DETAIL_TIME	PCF_FILE_TYPE	int	null
	PGE_ID	varchar(17)	not null
	PROCESSING_BOUNDARY	varchar(20)	null
PRODUCTION_REQUEST_PARAMETERS	PROCESSING_PERIOD	varchar(20)	null
	PRODUCTION_REQUEST_ID	varchar(20)	not null
	PGE_ID	varchar(17)	not null
	PGE_PARAMETER_LOGICAL_ID	int	not null
DPR_COMPLETION	PGE_PARAMETER_VALUE	varchar(30)	null
	DPR_ID	varchar(27)	not null
	COMPLETION_DATE	datetime	not null
	DPR_ELAPSED_TIME	float	not null
	PGE_ELAPSED_TIME	float	not null
	PGE_BLOCK_INPUT_OPERATIONS	int	null
	PGE_BLOCK_OUTPUT_OPERATIONS	int	null
	PGE_SWAPS	int	null
	PGE_MAX_MEMORY_USE	float	null
	PGE_SHARED_MEMORY_USE	float	null
	PGE_CPU_TIME	float	null
	SYSTEM_CPU_TIME	float	null
	PGE_PAGE_FAULTS	int	null
ESDT_PARM_VALUES	EXIT_CODE	int	null
	PLATFORM	varchar(60)	null
	DATA_TYPE_INSTANCE_ID	varchar(30)	not null
	ESDT_PARM_NAME	varchar(40)	not null
	ESDT_PARM_VAL	varchar(80)	null

METADATA_CHECKS	PGE_ID	varchar(17)	not null
	DATA_TYPE_ID	varchar(20)	not null
	PARA_NAME	varchar(40)	not null
	TYPE	varchar(5)	null
	OPERATOR	varchar(3)	null
	VALUE	varchar(40)	null
RSC_CPU_RAM_ALLOCATION	CPU_RAM_ALLOCATION_JOB_ID	varchar(40)	not null
	COMPUTER_ID	int	not null
	CPU_RAM_ALLOCATION_CPU_COUNT	int	null
	CPU_RAM_ALLOCATION_RAM_SIZE	float	null
RSC_DISK_ALLOCATION	DISK_ALLOCATION_ID	int	not null
	COMPUTER_ID	int	null
	DISK_PARTITION_ID	int	null
	DISK_ALLOCATION_TYPE	int	null
	DISK_ALLOCATION_PATH	varchar(240)	null
	DISK_ALLOCATION_SIZE	float	null
	DISK_ALLOCATION_USER	varchar(40)	null
	DISK_ALLOCATION_ACTUAL	float	null
RSV_RSC_EXPLOSION	RESERVATION_ID	int	not null
	RESOURCE_ID	int	not null
PCF	DPR_ID	varchar(30)	not null
	PCF_TYPE	int	not null
	PCF_NAME	varchar(60)	not null
	PCF_LOCATION	varchar(240)	not null
	PCF_PERMISSION	char(5)	not null
DP_ENUM_VALUES	ENUM_TYPE_ID	int	not null
	ENUM_NUMBER	int	not null
	ENUM_VALUE	char(10)	null
PGE_EXIT_DEPENDENCY	PGE_SSW_ID	varchar(17)	null
	DEPENDENCY_SSW_ID	varchar(17)	null
	EXIT_OPERATION	varchar(3)	null
	EXIT_CODE	int	null
PGE_EXIT_MESSAGE	PGE_SSW_ID	varchar(17)	null
	EXIT_CODE	int	null
	EXIT_MESSAGE	varchar(240)	null
ESDT_PARM_NAME	DATA_TYPE_ID	varchar(20)	not null
	ESDT_PARM_NAME	varchar(40)	not null
RESERVATION_INTERVAL	RESERVATION_ID	int	not null
	START_TIME	datetime	not null
	STOP_TIME	datetime	null
	FLAG	int default 0 between 0 & 1	null
PLAN_PR_EXPLOSION	PLAN_ID	varchar(20)	not null
	PRODUCTION_REQUEST_ID	varchar(20)	not null
	PRIORITY	int	null

9.3 Scripts

9.3.1 General

9.3.2 PDPS_PHASE3_SCHEMA.SQL

This script has three major functions:

- 1) to remove all old pdps_db_ops tables if they are exist
- 2) to create the new pdps_db_ops tables and indexs
- 3) to set up the tables referential integrity constraints

9.3.3 GRANT_PDPS_PHASE3

This script is to grant permission to the testbed users. The testbed users accounts must be established and added into the pdps_db_ops database before this script is executed.

9.3.4 Initialization

The following three scripts initialized the pdps_db_ops database before this database can be used:

- load_activities.sql
- load_dataserver_resource.sql
- loadmessages.sql

9.3.5 Database Management and Recovery

- scripts to run the database consistency checker (dbcc); run before database dump
- scripts to perform complete database dumps
- scripts to perform database transaction log dumps
- scripts to monitor the disk space available on the device(s) to which databases and trans logs are dumped
- scripts to monitor the space used by each database

9.3.6 SQL Server Maintenance

- scripts to detect the presence of error messages in the SQL Server and Backup Server errorlogs
- scripts to monitor OS files (e.g., /var/adm/messages on HP, etc.) where messages indicating OS problem that may lead to database corruption
- scripts to boot each SQL Server and all related COTS and/or client software
- scripts to kill the SQL Server and all related COTS and/or/client software
- scripts to establish named caches and buffer pools (for SQL Server System 11 or higher)
- scripts to monitor the size of the SQL Server errorlog and to prune it when needed

- scripts used to update statistics of the indexes on database tables á documented commands used to initialize SQL Servers and create and configure databases and database objects:disk init
 - create database
 - alter database
 - sp_addsegment
 - create table
 - create index
 - names, locations and sizes of all devices or files initialized as database devices
 - scripts used to add SQL Server logins
 - scripts used to add SQL Server databases users logins

9.4 Stored Procedures

The following trigger store procedures are installed on the autosys database. These triggers will be used by the autosys table to notify the pdps_db_ops database about the job status.

The following stored procedures are installed on pdps_db_ops database:

- procGetComments.sql
- procGetResources.sql
- procInsertGroundEvent.sql
- proc_u_DPR_COMPLETION.sql
- proc_u_DPR_COMPLETION_STATE.sql
- proc_u_DPR_PRIORITY.sql

The following trigger stored procedures are installed on the autosys database. These triggers will be used by the autosys table to notify the pdps_db_ops database about the job status.

- trg_u_JOB.sql
- trg_u_JOB_STATUS.sql

This page intentionally left blank.